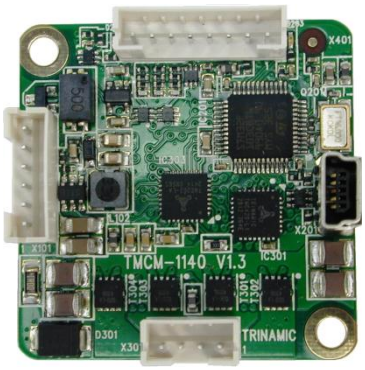
## Firmware Version V1.27

# TMCL™ FIRMWARE MANUAL




## TMCM-1140

1-Axis Stepper
Controller / Driver
2 A / 24 V
sensOstep™ Encoder
USB, RS485, and CAN

**UNIQUE FEATURES:**

coolStep™

stallGuard2™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

**www.trinamic.com**

TRINAMIC
MOTION CONTROL

# Table of Contents

1 Features ... 4
2 Putting the Module into Operation ... 6
2.1 Basic Set-Up ... 6
2.1.1 Start the TMCL-IDE Software Development Environment ... 8
2.2 Using TMCL Direct Mode ... 9
2.2.1 Important Motor Settings ... 10
2.3 Testing with a Simple TMCL Program ... 11
3 TMCL and the TMCL-IDE: Introduction ... 12
3.1 Binary Command Format ... 12
3.1.1 Checksum Calculation ... 13
3.2 Reply Format ... 13
3.2.1 Status Codes ... 14
3.3 Standalone Applications ... 14
3.4 TMCL Command Overview ... 15
3.4.1 TMCL Commands ... 15
3.4.2 Commands Listed According to Subject Area ... 16
3.5 The ASCII Interface ... 20
3.5.1 Format of the Command Line ... 20
3.5.2 Format of a Reply ... 20
3.5.3 Configuring the ASCII Interface ... 21
3.6 Commands ... 22
3.6.1 ROR (rotate right) ... 22
3.6.2 ROL (rotate left) ... 23
3.6.3 MST (motor stop) ... 24
3.6.4 MVP (move to position) ... 25
3.6.5 SAP (set axis parameter) ... 27
3.6.6 GAP (get axis parameter) ... 28
3.6.7 STAP (store axis parameter) ... 29
3.6.8 RSAP (restore axis parameter) ... 30
3.6.9 SGP (set global parameter) ... 31
3.6.10 GGP (get global parameter) ... 32
3.6.11 STGP (store global parameter) ... 33
3.6.12 RSGP (restore global parameter) ... 34
3.6.13 RFS (reference search) ... 35
3.6.14 SIO (set input / output) ... 36
3.6.15 GIO (get input /output) ... 38
3.6.16 CALC (calculate) ... 40
3.6.17 COMP (compare) ... 41
3.6.18 JC (jump conditional) ... 42
3.6.19 JA (jump always) ... 43
3.6.20 CSUB (call subroutine) ... 44
3.6.21 RSUB (return from subroutine) ... 45
3.6.22 WAIT (wait for an event to occur) ... 46
3.6.23 STOP (stop TMCL program execution) ... 47
3.6.24 SCO (set coordinate) ... 48
3.6.25 GCO (get coordinate) ... 49
3.6.26 CCO (capture coordinate) ... 50
3.6.27 ACO (accu to coordinate) ... 51
3.6.28 CALCX (calculate using the X register) ... 52
3.6.29 AAP (accumulator to axis parameter) ... 53
3.6.30 AGP (accumulator to global parameter) ... 54
3.6.31 CLE (clear error flags) ... 55
3.6.32 VECT (set interrupt vector) ... 56
3.6.33 EI (enable interrupt) ... 57
3.6.34 DI (disable interrupt) ... 58
3.6.35 RETI (return from interrupt) ... 59

3.6.36 Customer Specific TMCL Command Extension (UF0… UF7 / User Function) ..... 60
3.6.37 Request Target Position Reached Event ..... 60
3.6.38 TMCL Control Functions ..... 61
4 Axis Parameters ..... 63
4.1 stallGuard2 ..... 70
4.2 coolStep Related Axis Parameters ..... 70
4.3 Reference Search ..... 72
4.3.1 Reference Search Modes (Axis Parameter 193) ..... 73
4.4 Changing the Prescaler Value of an Encoder ..... 76
5 Global Parameters ..... 78
5.1 Bank 0 ..... 78
5.2 Bank 1 ..... 81
5.3 Bank 2 ..... 81
5.4 Bank 3 ..... 82
6 TMCL Programming Techniques and Structure ..... 83
6.1 Initialization ..... 83
6.2 Main Loop ..... 83
6.3 Using Symbolic Constants ..... 83
6.4 Using Variables ..... 84
6.5 Using Subroutines ..... 84
6.6 Mixing Direct Mode and Standalone Mode ..... 85
7 Life Support Policy ..... 86
8 Revision History ..... 87
8.1 Firmware Revision ..... 87
8.2 Document Revision ..... 87
9 References ..... 88

# 1 Features

The TMCM-1140 is a single axis controller/driver module for 2-phase bipolar stepper motors with state of the art feature set. It is highly integrated, offers a convenient handling and can be used in many decentralized applications. The module can be mounted on the back of NEMA 17 (42mm flange size) stepper motors and has been designed for coil currents up to 2 A RMS and 24 V DC supply voltage. With its high energy efficiency from TRINAMIC's coolStep™ technology cost for power consumption is kept down. The TMCL™ firmware allows for both, standalone operation and direct mode.

## MAIN CHARACTERISTICS

**Motion controller**

- Motion profile calculation in real-time
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- High performance microcontroller for overall system control and serial communication protocol handling

**Bipolar stepper motor driver**

- Up to 256 microsteps per full step
- High-efficient operation, low power dissipation
- Dynamic current control
- Integrated protection
- stallGuard2 feature for stall detection
- coolStep feature for reduced power consumption and heat dissipation

**Encoder**

- sensOstep magnetic encoder (1024 increments per rotation) e.g. for step-loss detection under all operating conditions and positioning supervision

**Interfaces**

- RS485 2-wire communication interface
- CAN 2.0B communication interface
- USB full speed (12Mbit/s) device interface
- 4 multipurpose inputs:
  - 3x general-purpose digital inputs
    (Alternate functions: STOP_L / STOP_R / HOME switch inputs or A/B/N encoder input)
  - 1x dedicated analog input
- 2 general purpose outputs
  - 1x open-drain 1A max.
  - 1x +5V supply output (can be switched on/off in software)

**Software**

- TMCL: standalone operation or remote controlled operation,
  program memory (non volatile) for up to 2048 TMCL commands, and
  PC-based application development software TMCL-IDE available for free.

**Electrical and mechanical data**

- Supply voltage: +24 V DC nominal (9... 28 V DC)
- Motor current: up to 2 A RMS / 2.8 A peak (programmable)

***Refer to separate Hardware Manual, too.***

## TRINAMICs Unique Features – Easy to Use with TMCL

**stallGuard2™** stallGuard2 is a high-precision sensorless load measurement using the back EMF on the coils. It can be used for stall detection as well as other uses at loads below those which stall the motor. The stallGuard2 measurement value changes linearly over a wide range of load, velocity, and current settings. At maximum motor load, the value goes to zero or near to zero. This is the most energy-efficient point of operation for the motor.

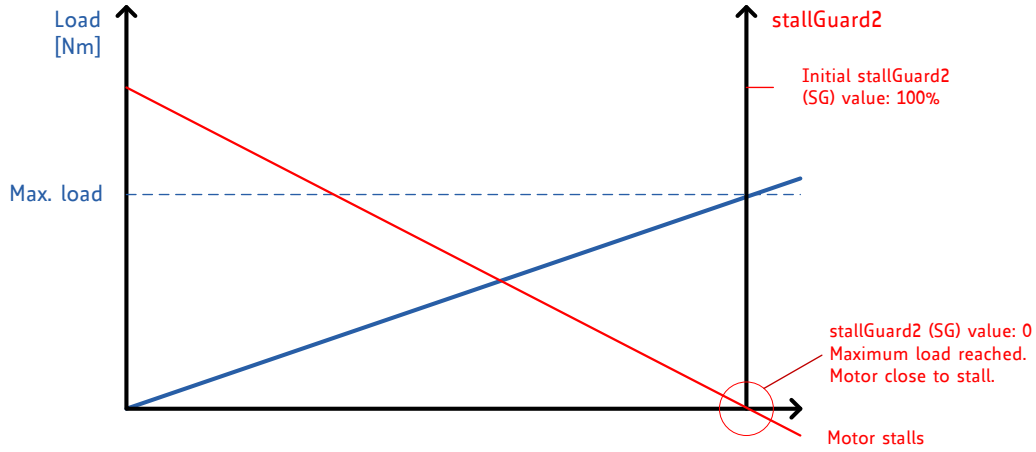



**Figure 1.1 stallGuard2 load measurement SG as a function of load**

**coolStep™** coolStep is a load-adaptive automatic current scaling based on the load measurement via stallGuard2 adapting the required current to the load. Energy consumption can be reduced by as much as 75%. coolStep allows substantial energy savings, especially for motors which see varying loads or operate at a high duty cycle. Because a stepper motor application needs to work with a torque reserve of 30% to 50%, even a constant-load application allows significant energy savings because coolStep automatically enables torque reserve when required. Reducing power consumption keeps the system cooler, increases motor life, and allows reducing cost.

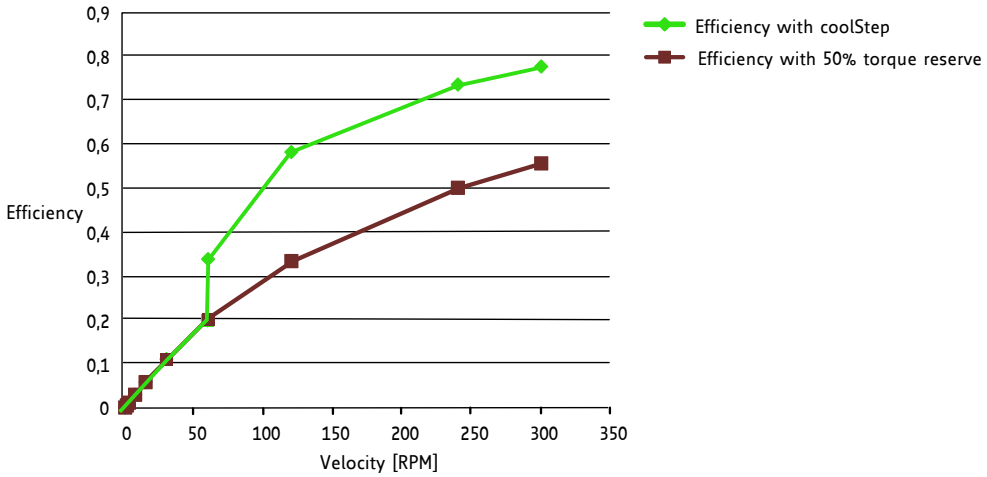



**Figure 1.2 Energy efficiency example with coolStep**

# 2 Putting the Module into Operation

Here you can find basic information for putting your TMCM-1140 into operation. If you are already common with TRINAMICs modules you may skip this chapter.

The things you need:

- TMCM-1140
- Interface (RS485/CAN/USB) suitable to your module with cables
- Nominal supply voltage +24V DC for your module
- TMCL-IDE program and PC
- Stepper motor

**PRECAUTIONS**

***Do not connect or disconnect the TMCM-1140 while powered!***
***Do not connect or disconnect the motor while powered!***
***Do not exceed the maximum power supply voltage of 28 V DC!***
***Note, that the module is not protected against reverse polarity!***
***START WITH POWER SUPPLY OFF!***

## 2.1 Basic Set-Up

The following paragraph will guide you through the steps of connecting the unit and making first movements with the motor.

### CONNECTING THE MODULE

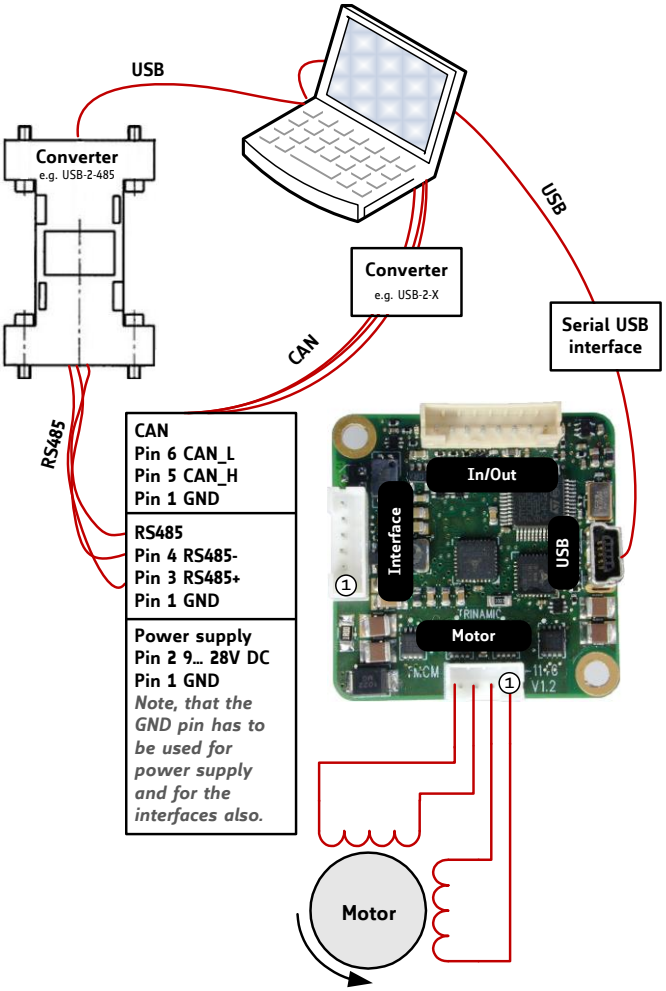



**Figure 2.1: Starting up**

1. **Connect power supply and choose your interface**

   a) **Connect CAN or RS485 and power supply**

   *CAN interface will be de-activated in case USB is connected due to internal sharing of hardware resources.*

| Pin | Label | Description |
|---|---|---|
| 1 | GND | System and signal ground |
| 2 | VDD | VDD (+9V…+28V) |
| 3 | RS485+ | RS485 interface, diff. signal (non-inverting) |
| 4 | RS485- | RS485 interface, diff. signal (inverting) |
| 5 | CAN_H | CAN interface, diff. signal (non-inverting) |
| 6 | CAN_L | CAN interface, diff. signal (inverting) |

   b) **Connect USB interface (as alternative to CAN and RS485; use a normal USB cable)**
   Download and install the file *TMCM-1140.inf* (www.trinamic.com).

| Pin | Label | Description |
|---|---|---|
| 1 | VBUS | +5V power |
| 2 | D- | Data – |
| 3 | D+ | Data + |
| 4 | ID | ground |
| 5 | GND | ground |

2. **Connect In/Out connector**
   If you like to work with the GPIOs or switches, use the In/Out connector.

| Pin | Label | Description |
|---|---|---|
| 1 | GND | System and signal ground |
| 2 | VDD | VDD, connected to VDD pin of the power and communication connector |
| 3 | OUT_1 | Open-drain output (max. 1A)<br>Integrated freewheeling diode to VDD |
| 4 | OUT_0 | +5V supply output (max. 100mA)<br>Can be switched on/off in software |
| 5 | AIN_0 | Dedicated analog input,<br>Input voltage range: 0..+10V<br>Resolution: 12bit (0..4095) |
| 6 | IN_0,<br>STOP_L,<br>ENC_A | General purpose digital input (+24V compatible)<br>Alternate function 1: left stop switch input<br>Alternate function 2: external incremental encoder channel A input |
| 7 | IN_1,<br>STOP_R,<br>ENC_B | General purpose digital input (+24V compatible)<br>Alternate function 1: right stop switch input<br>Alternate function 2: external incremental encoder channel B input |
| 8 | IN_2,<br>HOME,<br>ENC_N | General purpose digital input (+24V compatible)<br>Alternate function 1: home switch input<br>Alternate function 2: external incremental encoder index / zero channel input |

3. **Connect the motor**

| Pin | Label | Description |
|---|---|---|
| 1 | OB2 | Pin 2 of motor coil B |
| 2 | OB1 | Pin 1 of motor coil B |
| 3 | OA2 | Pin 2 of motor coil A |
| 4 | OA1 | Pin 1 of motor coil A |

4. **Switch ON the power supply**
   Turn power ON. The green LED for power lights up and the motor is powered but in standstill now.

   ***If this does not occur, switch power OFF and check your connections as well as the power supply.***

### 2.1.1 Start the TMCL-IDE Software Development Environment

The TMCL-IDE is available on www.trinamic.com.

Installing the TMCL-IDE:
Make sure the COM port you intend to use is not blocked by another program.
Open TMCL-IDE by clicking ***TMCL.exe***.
Choose ***Setup*** and ***Options*** and thereafter the ***Connection tab***.
Choose ***COM port*** and ***type*** with the parameters shown in Figure 2.2 (baud rate 9600). Click ***OK***.

USB interface
If the file *TMCM-1140.inf* is installed correctly, the module will be identified automatically.

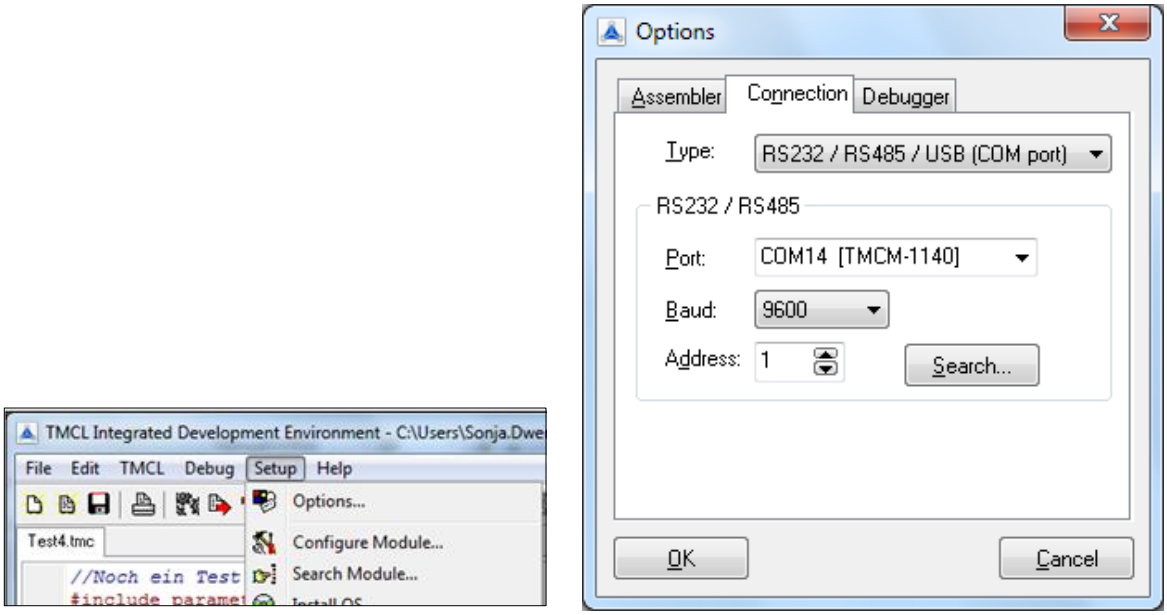
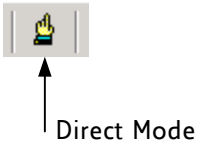



**Figure 2.2 Setup dialogue and connection tab of the TMCL-IDE.**

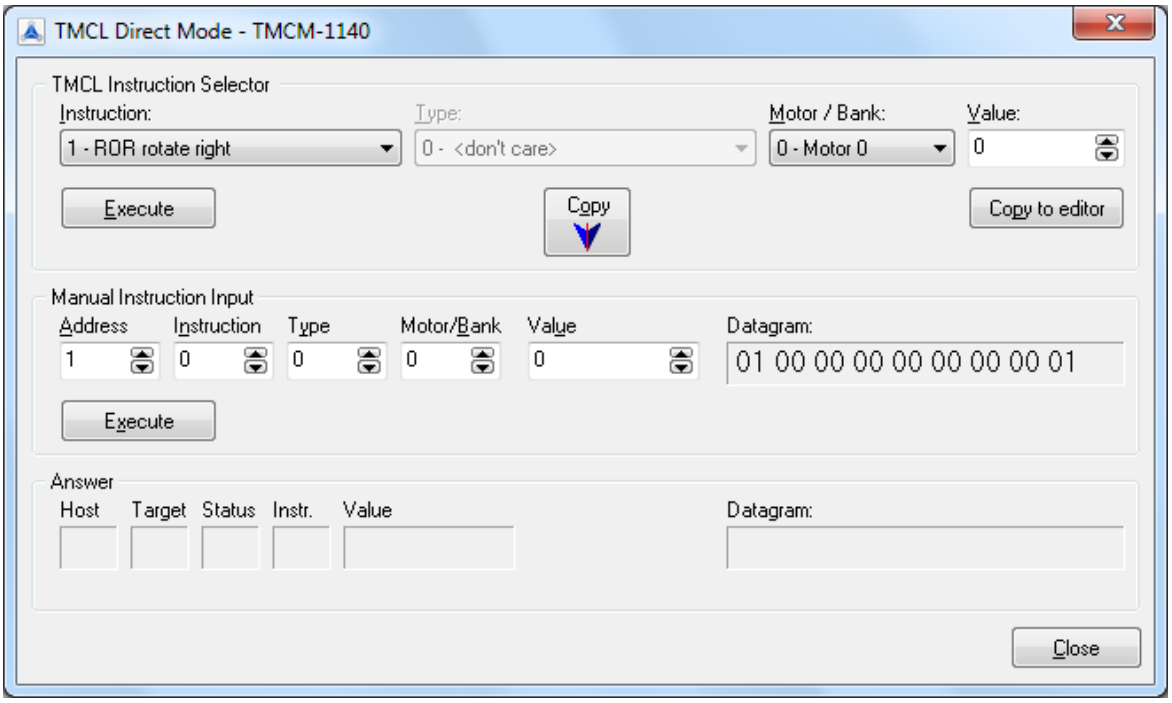
***Please refer to the TMCL-IDE User Manual for more information (see www.TRINAMIC.com).***

## 2.2 Using TMCL Direct Mode

1. Start TMCL ***Direct Mode***.




2. If the communication is established the TMCM-1140 is automatically detected. ***If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).***




3. Issue a command by choosing ***Instruction***, ***Type*** (if necessary), ***Motor***, and ***Value*** and click ***Execute*** to send it to the module.

Examples:

- ROR rotate right, motor 0, value 500 -> Click *Execute*. The motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The motor stops now.

Top right of the *TMCL Direct Mode* window is the button *Copy to editor*. Click here to copy the chosen command and create your own TMCL program. The command will be shown immediately on the editor.

*Note*:
Chapter 4 of this manual (axis parameters) includes a diagram which points out the coolStep related axis parameters and their functions.

### 2.2.1 Important Motor Settings

There are some axis parameters which have to be adjusted right in the beginning after installing your module. Please set the upper limiting values for the speed (axis parameter 4), the acceleration (axis parameter 5), and the current (axis parameter 6). Further set the standby current (axis parameter 7) and choose your microstep resolution with axis parameter 140. Please use the *SAP* (Set Axis Parameter) command for adjusting these values. The SAP command is described in paragraph 3.6.5. You can use the TMCL-IDE direct mode for easily configuring your module.

*Attention*:
The most important motor setting is the *absolute maximum motor current* setting, since too high values might cause motor damage!

**IMPORTANT AXIS PARAMETERS FOR MOTOR SETTING**

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 4 | Maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 5 | Maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 429 datasheet for calculation of physical units. | 0… 2047*[1] |
| 6 | Absolute max. current (CS / Current Scale) | The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps.<br>0… 7 \| 79…87 \| 160… 167 \| 240… 247<br>8… 15 \| 88… 95 \| 168… 175 \| 248… 255<br>16… 23 \| 96… 103 \| 176… 183<br>24… 31 \| 104… 111 \| 184… 191<br>32… 39 \| 112… 119 \| 192… 199<br>40… 47 \| 120… 127 \| 200… 207<br>48… 55 \| 128… 135 \| 208… 215<br>56… 63 \| 136… 143 \| 216… 223<br>64… 71 \| 144… 151 \| 224… 231<br>72… 79 \| 152… 159 \| 232… 239<br>*The most important motor setting, since too high values might cause motor damage!* | 0… 255<br>$I_{peak} =< value >\times \frac{2.8A}{255}$<br>$I_{RMS} =< value >\times \frac{2A}{255}$ |
| 7 | Standby current | The current limit two seconds after the motor has stopped. | 0… 255<br>$I_{peak} =< value >\times \frac{2.8A}{255}$<br>$I_{RMS} =< value >\times \frac{2A}{255}$ |
| 140 | Microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | 0… 8 |

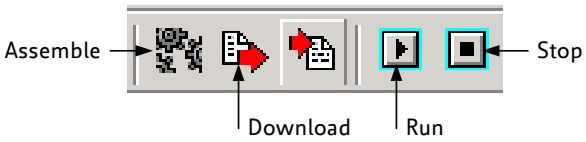
*[1] Unit of acceleration: $\frac{16MHz^2}{536870912 \cdot 2^{puls\_divisor+ramp\_divisor}} \frac{\text{microsteps}}{\text{sec}^2}$

## 2.3 Testing with a Simple TMCL Program

Type in the following program:

```
        ROL 0, 500                  //Rotate motor 0 with speed 10000
        WAIT TICKS, 0, 500
        MST 0
        ROR 0, 500                   //Rotate motor 0 with 50000
        WAIT TICKS, 0, 500
        MST 0

        SAP 4, 0, 500                 //Set max. Velocity
        SAP 5, 0, 50                  //Set max. Acceleration
Loop:   MVP ABS, 0, 10000             //Move to Position 10000
        WAIT POS, 0, 0                //Wait until position reached
        MVP ABS, 0, -10000            //Move to Position -10000
        WAIT POS, 0, 0                //Wait until position reached
        JA Loop                       //Infinite Loop
```




1. Click the ***Assemble*** icon to convert the TMCL program into binary code.
2. Then download the program to the TMCM-1140 module by clicking the ***Download*** icon.
3. Click the ***Run*** icon. The desired program will be executed.
4. Click the ***Stop*** button to stop the program.

# 3 TMCL and the TMCL-IDE: Introduction

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1140 consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (http://www.trinamic.com).

The TMCM-1140 supports TMCL direct mode (binary commands) and standalone TMCL program execution. You can store up to 2048 TMCL instructions on it. In direct mode and most cases the TMCL communication over RS485, CAN, or USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-1140. The TMCL interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485/CAN/USB to the bus master. Only then should the master transfer the next command. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.

The Trinamic Motion Control Language [TMCL] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).

Every command has a binary representation and a mnemonic. The binary format is used to send commands from the host to a module in direct mode, whereas the mnemonic format is used for easy usage of the commands when developing standalone TMCL applications using the TMCL-IDE (IDE means *Integrated Development Environment*).

There is also a set of configuration variables for the axis and for global parameters which allow individual configuration of nearly every function of a module. This manual gives a detailed description of all TMCL commands and their usage.

## 3.1 Binary Command Format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS485 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.

This is different when communicating is via the CAN bus. Address and checksum are included in the CAN standard and do not have to be supplied by the user.

The binary command format for R485/USB is as follows:

| Bytes | Meaning |
|---|---|
| 1 | Module address |
| 1 | Command number |
| 1 | Type number |
| 1 | Motor or Bank number |
| 4 | Value (MSB first!) |
| 1 | Checksum |

- The checksum is calculated by adding up all the other bytes using an 8-bit addition.
- When using CAN bus, just leave out the first byte (module address) and the last byte (checksum).

### 3.1.1 Checksum Calculation

As mentioned above, the checksum is calculated by adding up all bytes (including the module address byte) using 8-bit addition. Here are two examples to show how to do this:

```
- in C:
unsigned char i, Checksum;
unsigned char Command[9];

//Set the "Command" array to the desired command
Checksum = Command[0];
for(i=1; i<8; i++)
   Checksum+=Command[i];

Command[8]=Checksum; //insert checksum as last byte of the command
//Now, send it to the module

- in Delphi:
var
  i, Checksum: byte;
  Command: array[0..8] of byte;

  //Set the "Command" array to the desired command

  //Calculate the Checksum:
  Checksum:=Command[0];
  for i:=1 to 7 do Checksum:=Checksum+Command[i];
  Command[8]:=Checksum;
  //Now, send the "Command" array (9 bytes) to the module
```

## 3.2 Reply Format

Every time a command has been sent to a module, the module sends a reply.

The reply format for RS485/ /USB is as follows:

| Bytes | Meaning |
|---|---|
| 1 | Reply address |
| 1 | Module address |
| 1 | Status (e.g. 100 means *no error*) |
| 1 | Command number |
| 4 | Value (MSB first!) |
| 1 | Checksum |

- The checksum is also calculated by adding up all the other bytes using an 8-bit addition.
- When using CAN bus, just leave out the first byte (module address) and the last byte (checksum).
- Do not send the next command before you have received the reply!

### 3.2.1 Status Codes

The reply contains a status code. The status code can have one of the following values:

| Code | Meaning |
|---|---|
| 100 | Successfully executed, no error |
| 101 | Command loaded into TMCL program EEPROM |
| 1 | Wrong checksum |
| 2 | Invalid command |
| 3 | Wrong type |
| 4 | Invalid value |
| 5 | Configuration EEPROM locked |
| 6 | Command not available |

## 3.3 Standalone Applications

The module is equipped with a TMCL memory for storing TMCL applications. You can use TMCL-IDE for developing standalone TMCL applications. You can download a program into the EEPROM and afterwards it will run on the module. The TMCL-IDE contains an editor and the TMCL assembler where the commands can be entered using their mnemonic format. They will be assembled automatically into their binary representations. Afterwards this code can be downloaded into the module to be executed there.

## 3.4 TMCL Command Overview

In this section a short overview of the TMCL commands is given.

### 3.4.1 TMCL Commands

| Command | Number | Parameter | Description |
|---|---|---|---|
| ROR | 1 | <motor number>, <velocity> | Rotate right with specified velocity |
| ROL | 2 | <motor number>, <velocity> | Rotate left with specified velocity |
| MST | 3 | <motor number> | Stop motor movement |
| MVP | 4 | ABS\|REL\|COORD, <motor number>, <position\|offset> | Move to position (absolute or relative) |
| SAP | 5 | <parameter>, <motor number>, <value> | Set axis parameter (motion control specific settings) |
| GAP | 6 | <parameter>, <motor number> | Get axis parameter (read out motion control specific settings) |
| STAP | 7 | <parameter>, <motor number> | Store axis parameter permanently (non volatile) |
| RSAP | 8 | <parameter>, <motor number> | Restore axis parameter |
| SGP | 9 | <parameter>, <bank number>, value | Set global parameter (module specific settings e.g. communication settings or TMCL™ user variables) |
| GGP | 10 | <parameter>, <bank number> | Get global parameter (read out module specific settings e.g. communication settings or TMCL™ user variables) |
| STGP | 11 | <parameter>, <bank number> | Store global parameter (TMCL™ user variables only) |
| RSGP | 12 | <parameter>, <bank number> | Restore global parameter (TMCL™ user variable only) |
| RFS | 13 | START\|STOP\|STATUS, <motor number> | Reference search |
| SIO | 14 | <port number>, <bank number>, <value> | Set digital output to specified value |
| GIO | 15 | <port number>, <bank number> | Get value of analogue/digital input |
| CALC | 19 | <operation>, <value> | Process accumulator & value |
| COMP | 20 | <value> | Compare accumulator <-> value |
| JC | 21 | <condition>, <jump address> | Jump conditional |
| JA | 22 | <jump address> | Jump absolute |
| CSUB | 23 | <subroutine address> | Call subroutine |
| RSUB | 24 | | Return from subroutine |
| EI | 25 | <interrupt number> | Enable interrupt |
| DI | 26 | <interrupt number> | Disable interrupt |
| WAIT | 27 | <condition>, <motor number>, <ticks> | Wait with further program execution |
| STOP | 28 | | Stop program execution |
| SCO | 30 | <coordinate number>, <motor number>, <position> | Set coordinate |
| GCO | 31 | <coordinate number>, <motor number> | Get coordinate |
| CCO | 32 | <coordinate number>, <motor number> | Capture coordinate |
| CALCX | 33 | <operation> | Process accumulator & X-register |
| AAP | 34 | <parameter>, <motor number> | Accumulator to axis parameter |
| AGP | 35 | <parameter>, <bank number> | Accumulator to global parameter |
| VECT | 37 | <interrupt number>, <label> | Set interrupt vector |
| RETI | 38 | | Return from interrupt |
| ACO | 39 | <coordinate number>, <motor number> | Accu to coordinate |

## 3.4.2 Commands Listed According to Subject Area

### 3.4.2.1 Motion Commands

These commands control the motion of the motor. They are the most important commands and can be used in direct mode or in standalone mode.

| Mnemonic | Command number | Meaning |
|---|---|---|
| ROL | 2 | Rotate left |
| ROR | 1 | Rotate right |
| MVP | 4 | Move to position |
| MST | 3 | Motor stop |
| RFS | 13 | Reference search |
| SCO | 30 | Store coordinate |
| CCO | 32 | Capture coordinate |
| GCO | 31 | Get coordinate |

### 3.4.2.2 Parameter Commands

These commands are used to set, read and store axis parameters or global parameters. Axis parameters can be set independently for each axis, whereas global parameters control the behavior of the module itself. These commands can also be used in direct mode and in standalone mode.

| Mnemonic | Command number | Meaning |
|---|---|---|
| SAP | 5 | Set axis parameter |
| GAP | 6 | Get axis parameter |
| STAP | 7 | Store axis parameter into EEPROM |
| RSAP | 8 | Restore axis parameter from EEPROM |
| SGP | 9 | Set global parameter |
| GGP | 10 | Get global parameter |
| STGP | 11 | Store global parameter into EEPROM |
| RSGP | 12 | Restore global parameter from EEPROM |

### 3.4.2.3 Control Commands

These commands are used to control the program flow (loops, conditions, jumps etc.). It does not make sense to use them in direct mode. They are intended for standalone mode only.

| Mnemonic | Command number | Meaning |
|---|---|---|
| JA | 22 | Jump always |
| JC | 21 | Jump conditional |
| COMP | 20 | Compare accumulator with constant value |
| CSUB | 23 | Call subroutine |
| RSUB | 24 | Return from subroutine |
| WAIT | 27 | Wait for a specified event |
| STOP | 28 | End of a TMCL™ program |

### 3.4.2.4 I/O Port Commands

These commands control the external I/O ports and can be used in direct mode and in standalone mode.

| Mnemonic | Command number | Meaning |
|---|---|---|
| SIO | 14 | Set output |
| GIO | 15 | Get input |

#### 3.4.2.5 Calculation Commands

These commands are intended to be used for calculations within TMCL applications. Although they could also be used in direct mode it does not make much sense to do so.

| Mnemonic | Command number | Meaning |
|---|---|---|
| CALC | 19 | Calculate using the accumulator and a constant value |
| CALCX | 33 | Calculate using the accumulator and the X register |
| AAP | 34 | Copy accumulator to an axis parameter |
| AGP | 35 | Copy accumulator to a global parameter |
| ACO | 39 | Copy accu to coordinate |

For calculating purposes there is an accumulator (or accu or A register) and an X register. When executed in a TMCL program (in standalone mode), all TMCL commands that read a value store the result in the accumulator. The X register can be used as an additional memory when doing calculations. It can be loaded from the accumulator.

When a command that reads a value is executed in direct mode the accumulator will not be affected. This means that while a TMCL program is running on the module (standalone mode), a host can still send commands like GAP and GGP to the module (e.g. to query the actual position of the motor) without affecting the flow of the TMCL™ program running on the module.

#### 3.4.2.6 Interrupt Commands

Due to some customer requests, interrupt processing has been introduced in the TMCL firmware for ARM based modules.

| Mnemonic | Command number | Meaning |
|---|---|---|
| EI | 25 | Enable interrupt |
| DI | 26 | Disable interrupt |
| VECT | 37 | Set interrupt vector |
| RETI | 38 | Return from interrupt |

##### 3.4.2.6.1 Interrupt Types

There are many different interrupts in TMCL, like timer interrupts, stop switch interrupts, position reached interrupts, and input pin change interrupts. Each of these interrupts has its own interrupt vector. Each interrupt vector is identified by its interrupt number. Please use the TMCL included file *Interrupts.inc* for symbolic constants of the interrupt numbers.

##### 3.4.2.6.2 Interrupt Processing

When an interrupt occurs and this interrupt is enabled and a valid interrupt vector has been defined for that interrupt, the normal TMCL program flow will be interrupted and the interrupt handling routine will be called. Before an interrupt handling routine gets called, the context of the normal program will be saved automatically (i.e. accumulator register, X register, TMCL flags).

*There is no interrupt nesting, i.e. all other interrupts are disabled while an interrupt handling routine is being executed.*

On return from an interrupt handling routine, the context of the normal program will automatically be restored and the execution of the normal program will be continued.

#### 3.4.2.6.3 Interrupt Vectors

The following table shows all interrupt vectors that can be used.

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | (Target) position reached |
| 15 | Stall (stallGuard2) |
| 21 | Deviation |
| 27 | Stop left |
| 28 | Stop right |
| 39 | IN_0 change |
| 40 | IN_1 change |
| 41 | IN_2 change |
| 42 | IN_3 change |
| 255 | Global interrupts |

#### 3.4.2.6.4 Further Configuration of Interrupts

Some interrupts need further configuration (e.g. the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type>, 3, <value>). Please refer to the SGP command (paragraph 3.6.9) for further information about that.

#### 3.4.2.6.5 Using Interrupts in TMCL

For using an interrupt proceed as follows:

- Define an interrupt handling routine using the VECT command.
- If necessary, configure the interrupt using an SGP <type>, 3, <value> command.
- Enable the interrupt using an EI <interrupt> command.
- Globally enable interrupts using an EI 255 command.
- An interrupt handling routine must always end with a RETI command

**EXAMPLE FOR THE USE OF A TIMER INTERRUPT:**

```
    VECT 0, TimerOIrq  //define the interrupt vector
    SGP 0, 3, 1000     //configure the interrupt: set its period to 1000ms
    EI 0               //enable this interrupt
    EI 255             //globally switch on interrupt processing

//Main program: toggles output 3, using a WAIT command for the delay
Loop:
    SIO 3, 2, 1
    WAIT TICKS, 0, 50
    SIO 3, 2, 0
    WAIT TICKS, 0, 50
    JA Loop

//Here is the interrupt handling routine
TimerOIrq:
    GIO 0, 2           //check if OUT0 is high
    JC NZ, Out0Off     //jump if not
    SIO 0, 2, 1        //switch OUT0 high
    RETI               //end of interrupt
Out0Off:
    SIO 0, 2, 0        //switch OUT0 low
    RETI               //end of interrupt
```

In the example above, the interrupt numbers are used directly. To make the program better readable use the provided include file *Interrupts.inc*. This file defines symbolic constants for all interrupt numbers which can be used in all interrupt commands. The beginning of the program above then looks like the following:

```
#include Interrupts.inc
    VECT TI_TIMER0, Timer0Irq
    SGP TI_TIMER0, 3, 1000
    EI TI_TIMER0
    EI TI_GLOBAL
```

Please also take a look at the other example programs.

## 3.5 The ASCII Interface

There is also an ASCII interface that can be used to communicate with the module and to send some commands as text strings.

**THE FOLLOWING COMMANDS CAN BE USED IN ASCII MODE:**

ROL, ROR, MST, MVP, SAP, GAP, STAP, RSAP, SGP, GGP, STGP, RSGP, RFS, SIO, GIO, SCO, GCO, CCO, UF0, UF1, UF2, UF3, UF4, UF5, UF6, and UF7.

*Note*:
Only direct mode commands can be entered in ASCII mode!

**SPECIAL COMMANDS WHICH ARE ONLY AVAILABLE IN ASCII MODE:**

- BIN: This command quits ASCII mode and returns to binary TMCL™ mode.
- RUN: This command can be used to start a TMCL™ program in memory.
- STOP: Stops a running TMCL™ application.

**ENTERING AND LEAVING ASCII MODE:**

1. The ASCII command line interface is entered by sending the binary command 139 (*enter ASCII mode*).
2. Afterwards the commands are entered as in the TMCL-IDE.
3. For leaving the ASCII mode and re-enter the binary mode enter the command BIN.

### 3.5.1 Format of the Command Line

As the first character, the address character has to be sent. The address character is *A* when the module address is 1, *B* for modules with address 2 and so on. After the address character there may be spaces (but this is not necessary). Then, send the command with its parameters. At the end of a command line a <CR> character has to be sent.

**EXAMPLES FOR VALID COMMAND LINES:**

```
AMVP ABS, 1, 50000
A MVP ABS, 1, 50000
AROL 2, 500
A MST 1
ABIN
```

*The command lines above address the module with address 1. To address e.g. module 3, use address character C instead of A. The last command line shown above will make the module return to binary mode.*

### 3.5.2 Format of a Reply

After executing the command the module sends back a reply in ASCII format.

The reply consists of:

- the address character of the host (host address that can be set in the module)
- the address character of the module
- the status code as a decimal number
- the return value of the command as a decimal number
- a <CR> character

*So, after sending AGAP 0, 1 the reply would be BA 100 –5000 if the actual position of axis 1 is –5000, the host address is set to 2 and the module address is 1. The value 100 is the status code 100 that means command successfully executed.*

### 3.5.3 Configuring the ASCII Interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. ***Global parameter 67 is used for this purpose*** (please see also chapter 5.1).

Bit 0 determines the startup mode: if this bit is set, the module starts up in ASCII mode, else it will start up in binary mode (default).

Bit 4 and Bit 5 determine how the characters that are entered are echoed back. Normally, both bits are set to zero. In this case every character that is entered is echoed back when the module is addressed. Character can also be erased using the backspace character (press the backspace key in a terminal program).

When bit 4 is set and bit 5 is clear the characters that are entered are not echoed back immediately but the entire line will be echoed back after the <CR> character has been sent.

When bit 5 is set and bit 4 is clear there will be no echo, only the reply will be sent. This may be useful in RS485 systems.

# 3.6 Commands

The module specific commands are explained in more detail on the following pages. They are listed according to their command number.

## 3.6.1 ROR (rotate right)

With this command the motor will be instructed to rotate with a specified velocity in *right* direction (increasing the position counter).

**Internal function:** First, velocity mode is selected. Then, the velocity value is transferred to axis parameter #0 (*target velocity*).

The module is based on the TMC429 stepper motor controller and the TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROL, MST, SAP, GAP

**Mnemonic:** ROR 0, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 1 | (don't care) | 0* | <velocity><br>0... 2047 |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Rotate right, velocity = 350
*Mnemonic:* ROR 0, 350

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $01 | $00 | $00 | $00 | $00 | $01 | $5e |

### 3.6.2 ROL (rotate left)

With this command the motor will be instructed to rotate with a specified velocity (opposite direction compared to ROR, decreasing the position counter).

**Internal function:** First, velocity mode is selected. Then, the velocity value is transferred to axis parameter #0 (*target velocity*).

The module is based on the TMC429 stepper motor controller and the TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROR, MST, SAP, GAP

**Mnemonic:** ROL 0, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 2 | (don't care) | 0* | <velocity><br>0... 2047 |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Rotate left, velocity = 1200
*Mnemonic:* ROL 0, 1200

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $02 | $00 | $00 | $00 | $00 | $04 | $b0 |

### 3.6.3 MST (motor stop)

With this command the motor will be instructed to stop with a soft stop.

**Internal function:** The axis parameter *target velocity* is set to zero.

**Related commands:** ROL, ROR, SAP, GAP

**Mnemonic:** MST 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 3 | (don't care) | 0* | (don't care) |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Stop motor
*Mnemonic:* MST 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $03 | $00 | $00 | $00 | $00 | $00 | $00 |

### 3.6.4 MVP(move to position)

With this command the motor will be instructed to move to a specified relative or absolute position. It will use the acceleration/deceleration ramp and the positioning speed programmed into the unit. This command is non-blocking – that is, a reply will be sent immediately after command interpretation and initialization of the motion controller. Further commands may follow without waiting for the motor reaching its end position. The maximum velocity and acceleration are defined by axis parameters #4 and #5.

The range of the MVP command is 32 bit signed (-2.147.483.648... +2.147.483.647). Positioning can be interrupted using MST, ROL or ROR commands.

**THREE OPERATION TYPES ARE AVAILABLE:**

- Moving to an absolute position in the range from -2.147.483.648... +2.147.483.647 ($-2^{31}$... $2^{31}$-1).
- Starting a relative movement by means of an offset to the actual position. In this case, the new resulting position value must not exceed the above mentioned limits, too.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

*Please note, that the distance between the actual position and the new one should not be more than* 2.147.483.647 *(*$2^{31}$-1*) microsteps. Otherwise the motor will run in the opposite direction in order to take the shorter distance.*

**Internal function:** A new position value is transferred to the axis parameter #2 target position".

**Related commands:** SAP, GAP, SCO, CCO, GCO, MST

**Mnemonic:** MVP <ABS|REL|COORD>, 0, <position|offset|coordinate number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 4 | 0 ABS – absolute | 0* | <position> |
| | 1 REL – relative | 0 | <offset> |
| | 2 COORD – coordinate | 0 | <coordinate number> (0... 20) |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Move motor to (absolute) position 90000
*Mnemonic:* MVP ABS, 0, 9000

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $04 | $00 | $00 | $00 | $01 | $5f | $90 |

**Example:**

Move motor from current position 1000 steps backward (move relative –1000)
*Mnemonic:* MVP REL, 0, -1000

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $04 | $01 | $00 | $ff | $ff | $fc | $18 |

**Example:**

Move motor to previously stored coordinate #8
*Mnemonic:* MVP COORD, 0, 8

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $04 | $02 | $00 | $00 | $00 | $00 | $08 |

***When moving to a coordinate, the coordinate has to be set properly in advance with the help of the SCO, CCO or ACO command.***

### 3.6.5 SAP (set axis parameter)

With this command most of the motion control parameters of the module can be specified. The settings will be stored in SRAM and therefore are volatile. That is, information will be lost after power off.
***Please use command STAP (store axis parameter) in order to store any setting permanently.***

For a table with parameters and values which can be used together with this command please refer to chapter 4.

**Internal function:** The parameter format is converted ignoring leading zeros (or ones for negative values). The parameter is transferred to the correct position in the appropriate device.

**Related commands:** GAP, STAP, RSAP, AAP

**Mnemonic:** SAP <parameter number>, 0, <value>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 5 | <parameter number> | 0* | <value> |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Set the absolute maximum current of motor to 200mA
*Mnemonic:* SAP 6, 0, 200

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $05 | $06 | $00 | $00 | $00 | $00 | $c8 |

### 3.6.6 GAP (get axis parameter)

Most parameters of the TMCM-1140 can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only output in the *value* field of the reply (without affecting the accumulator).

For a table with parameters and values which can be used together with this command please refer to chapter 4.

**Internal function:** The parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

**Related commands:** SAP, STAP, AAP, RSAP

**Mnemonic:** GAP <parameter number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 6 | <parameter number> | 0* | (don't care) |

***motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Get the actual position of motor
*Mnemonic:* GAP 0, 1

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $06 | $01 | $00 | $00 | $00 | $00 | $00 |

*Reply:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Host-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $02 | $01 | $64 | $06 | $00 | $00 | $02 | $c7 |

⇨ **status=no error, position=711**

### 3.6.7 STAP (store axis parameter)

An axis parameter previously set with a *Set Axis Parameter* command (SAP) will be stored permanent. Most parameters are automatically restored after power up (refer to axis parameter list in chapter 4).

For a table with parameters and values which can be used together with this command please refer to chapter 4.

**Internal function:** An axis parameter value stored in SRAM will be transferred to EEPROM and loaded from EEPORM after next power up.

**Related commands:** SAP, RSAP, GAP, AAP

**Mnemonic:** STAP <parameter number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 7 | <parameter number> | 0*[1] | (don't care)*[2] |

***[1]motor number is always 0 as only one motor is involved***
***[2]the value operand of this function has no effect. Instead, the currently used value (e.g. selected by SAP) is saved.***

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Parameter ranges:**

| Parameter number | Motor number | Value |
|---|---|---|
| s. chapter 4 | 0 | s. chapter 4 |

**Example:**

Store the maximum speed of motor
*Mnemonic:* STAP 4, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $07 | $04 | $00 | $00 | $00 | $00 | $00 |

*Note*:
The STAP command will not have any effect when the configuration EEPROM is locked (refer to 5.1). In direct mode, the error code 5 (configuration EEPROM locked, see also section 3.2.1) will be returned in this case.

### 3.6.8 RSAP (restore axis parameter)

For all configuration-related axis parameters non-volatile memory locations are provided. By default, most parameters are automatically restored after power up (refer to axis parameter list in chapter 4). A single parameter that has been changed before can be reset by this instruction also.

For a table with parameters and values which can be used together with this command please refer to chapter 4.

**Internal function:** The specified parameter is copied from the configuration EEPROM memory to its RAM location.

**Relate commands:** SAP, STAP, GAP, and AAP

**Mnemonic:** RSAP <parameter number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 8 | <parameter number> | 0* | (don't care) |

***motor number is always 0 as only one motor is involved**

**Reply structure in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Restore the maximum current of motor
*Mnemonic:* RSAP 6, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $08 | $06 | $00 | $00 | $00 | $00 | $00 |

### 3.6.9 SGP (set global parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.

***All module settings will automatically be stored non-volatile (internal EEPROM of the processor). The TMCL user variables will not be stored in the EEPROM automatically, but this can be done by using STGP commands.***

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 5.

**Internal function:** the parameter format is converted ignoring leading zeros (or ones for negative values). The parameter is transferred to the correct position in the appropriate (on board) device.

**Related commands:** GGP, STGP, RSGP, AGP

**Mnemonic:** SGP <parameter number>, <bank number>, <value>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 9 | <parameter number> | <bank number> | <value> |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Set the serial address of the target device to 3
*Mnemonic:* SGP 66, 0, 3

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $09 | $42 | $00 | $00 | $00 | $00 | $03 |

## 3.6.10 GGP (get global parameter)

All global parameters can be read with this function. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 5.

**Internal function:** The parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

**Related commands:** SGP, STGP, RSGP, AGP

**Mnemonic:** GGP <parameter number>, <bank number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 10 | (see chapter 6) | <bank number> | (don't care) |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Get the serial address of the target device
*Mnemonic:* GGP 66, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $0a | $42 | $00 | $00 | $00 | $00 | $00 |

*Reply:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Host-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $02 | $01 | $64 | $0a | $00 | $00 | $00 | $01 |

⇨ **Status=no error, Value=1**

### 3.6.11 STGP (store global parameter)

This command is used to store TMCL user variables permanently in the EEPROM of the module. Some global parameters are located in RAM memory, so without storing modifications are lost at power down. This instruction enables enduring storing. Most parameters are automatically restored after power up.

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 5.

**Internal function:** The specified parameter is copied from its RAM location to the configuration EEPROM.

**Related commands:** SGP, GGP, RSGP, AGP

**Mnemonic:** STGP <parameter number>, <bank number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 11 | (see chapter 8) | <bank number> (see chapter 5) | (don't care) |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Store the user variable #42
*Mnemonic:* STGP 42, 2

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $0b | $2a | $02 | $00 | $00 | $00 | $00 |

### 3.6.12 RSGP (restore global parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. By default, most parameters are automatically restored after power up. A single parameter that has been changed before can be reset by this instruction.

**Internal function:** the specified parameter is copied from the configuration EEPROM memory to its RAM location.

**Relate commands:** SGP, STGP, GGP, and AGP

**Mnemonic:** RSAP <parameter number>, <bank number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 12 | <parameter number> | <bank number> | (don't care) |

**Reply structure in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 5.

**Example:**

Restore user variable #42
*Mnemonic:* RSGP 42, 2

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $0c | $2a | $02 | $00 | $00 | $00 | $00 |

## 3.6.13 RFS (reference search)

The TMCM-1140 has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and supports up-to three switches. The status of the reference search can also be queried to see if it has already finished. (In a TMCL program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 4 / 4.3). The reference search can be started, stopped, and the actual status of the reference search can be checked.

**Internal function:** the reference search is implemented as a state machine, so interaction is possible during execution.

**Related commands:** WAIT

**Mnemonic:** RFS <START|STOP|STATUS>, <motor>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 13 | 0 START – start ref. search<br>1 STOP – abort ref. search<br>2 STATUS – get status | 0 | see below |

**REPLY IN DIRECT MODE:**

When using type 0 (START) or 1 (STOP):

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

When using type 2 (STATUS):

| STATUS | VALUE | |
|---|---|---|
| 100 – OK | 0 | ref. search active |
| | other values | no ref. search active |

**Example:**

Start reference search of motor 0
*Mnemonic:* RFS START, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $0d | $00 | $00 | $00 | $00 | $00 | $00 |

*With this module it is possible to use stall detection instead of a reference search.*

## 3.6.14 SIO (set input / output)

- SIO sets the status of the two general purpose digital outputs either to zero (0) or to one (1). Bank 2 is used for this purpose.
- SIO is used to switch the pull-up resistors for all digital inputs ON (1) and OFF (0). Bank 0 is used for this purpose.

**Internal function:** the passed value is transferred to the specified output line.

**Related commands:** GIO, WAIT

**Mnemonic:** SIO <port number>, <bank number>, <value>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 14 | <port number> | <bank number> | <value><br>0/1 |

*Bank 2 is used for setting the status of the general digital output either to zero (0) or to one (1).*

**Reply structure:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Activate OUT_1, supply +5V to external circuits (bank 2, output 1)
*Mnemonic:* SIO 1, 2, 1

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $0e | $07 | $02 | $00 | $00 | $00 | $01 |

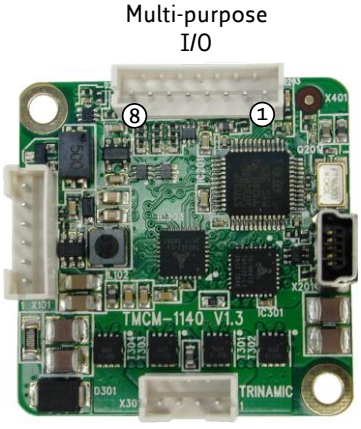



**Figure 3.1 I/O connector**

### I/O PORTS USED FOR SIO AND COMMAND

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 3 | OUT_0 | SIO 0, 2, <n> | 0 – OUT_0 off / high impedance / floating<br>1 – OUT_0 pulled low (max. 1A) |
| 4 | OUT_1 | SIO 1, 2, <n> | 0 – OUT_1 off / (weak) low via 10k pull-down resistor<br>1 – OUT_1 supplies +5V to external circuits (100mA load max.) |

#### ADDRESSING BOTH OUTPUT LINES WITH ONE SIO COMMAND

- Set the type parameter to 255 and the bank parameter to 2.
- The value parameter must then be set to a value between 0... 255, where every bit represents one output line.
- Furthermore, the value can also be set to -1. In this special case, the contents of the lower 8 bits of the accumulator are copied to the output pins.

**Example:**

Set all output pins high.
*Mnemonic:* SIO 255, 2, 3

#### THE FOLLOWING PROGRAM WILL SHOW THE STATES OF THE INPUT LINES ON THE OUTPUT LINES

```
Loop: GIO 255, 0
      SIO 255, 2,-1
      JA Loop
```

#### SPECIAL COMMAND FOR SWITCHING THE PULL-UP RESISTORS ON OR OFF FOR ALL THREE DIGITAL INPUTS AT ONCE

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 6 | IN_1 | SIO 0, 0, <n> | 1/0 |
| 7 | IN_2 | | 0: OFF |
| 8 | IN_3 | | 1: ON |

## 3.6.15 GIO (get input /output)

With this command the status of all general purpose inputs of the module can be read out. The function reads a digital or analog input port. Digital lines will read 0 and 1, while the ADC channels deliver their 12 bit result in the range of 0... 4095.

### GIO IN STANDALONE MODE

In standalone mode the requested value is copied to the *accumulator* (accu) for further processing purposes such as conditioned jumps.

### GIO IN DIRECT MODE

In direct mode the value is only output in the *value* field of the reply, without affecting the accumulator. The actual status of a digital output line can also be read.

**Internal function:** the specified line is read.

**Related commands:** SIO, WAIT

**Mnemonic:** GIO <port number>, <bank number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 15 | <port number> | <bank number> | don't care |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | <status of the port> |

**Example:**

Get the analog value of IN_0
*Mnemonic:* GIO 0, 1

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $0f | $00 | $01 | $00 | $00 | $00 | $00 |

*Reply:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Host-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $02 | $01 | $64 | $0f | $00 | $00 | $01 | $2e |

Status = no error, value = 320

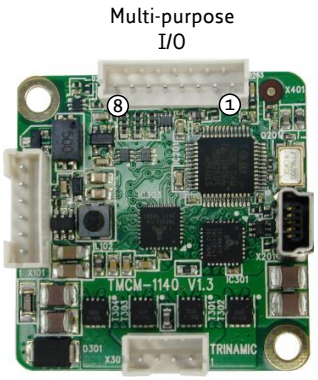



**Figure 3.2 I/O connector**

### 3.6.15.1 I/O Bank 0 – Digital Inputs

***The analog input IN_0 can be read as digital or analog input at the same time. The analog value of IN_0 can be accessed in bank 1.***

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 5 | IN_0 | GIO 0, 0 | 0/1 |
| 6 | IN_1 | GIO 1, 0 | 0/1 |
| 7 | IN_2 | GIO 2, 0 | 0/1 |
| 8 | IN_3 | GIO 3, 0 | 0/1 |

**ENC_N CHANNEL READ-OUT COMMAND**

| I/O port | Command |
|---|---|
| ENC_N channel input<br>0 off<br>1 active | GIO 11, 0 |

**READING ALL DIGITAL INPUTS WITH ONE GIO COMMAND**

- Set the type parameter to 255 and the bank parameter to 0.
- In this case the status of all digital input lines will be read to the lower eight bits of the accumulator.

**USE FOLLOWING PROGRAM TO REPRESENT THE STATES OF THE INPUT LINES ON THE OUTPUT LINES**

```
Loop: GIO 255, 0
      SIO 255, 2,-1
      JA Loop
```

*Note*: IN_0 can be used as analog or digital input.

### 3.6.15.2 I/O Bank 1 – Analog Input

***The analog input IN_0 can be read as digital or analog input at the same time. The analog value of IN_0 can be accessed in bank 1.***

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 5 | IN_0 | GIO 0, 1 | 0… 4095 |

**READING OUT OPERATING VOLTAGE AND TEMPERATURE:**

| I/O port | Command |
|---|---|
| Operating voltage [1/10 V] | GIO 8, 1 |
| Temperature [°C] | GIO 9, 1 |

#### 3.6.15.3 I/O Bank 2 –States of Digital Outputs

***The states of the OUT lines (that have been set by SIO commands) can be read back using bank 2.***

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 3 | OUT_0 | GIO 0, 2, <n> | 1/0 |
| 4 | OUT_1 | GIO 1, 2, <n> | 1/0 |

## 3.6.16 CALC (calculate)

A value in the accumulator variable, previously read by a function such as GAP (get axis parameter) can be modified with this instruction. Nine different arithmetic functions can be chosen and one constant operand value must be specified. The result is written back to the accumulator, for further processing like comparisons or data transfer.

**Related commands:** CALCX, COMP, JC, AAP, AGP, GAP, GGP

**Mnemonic:** CALC <operation>, <value>
where <op> is ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT or LOAD

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 19 | 0 ADD – add to accu<br>1 SUB – subtract from accu<br>1 MUL – multiply accu by<br>1 DIV – divide accu by<br>1 MOD – modulo divide by<br>1 AND – logical and accu with<br>6 OR – logical or accu with<br>7 XOR – logical exor accu with<br>8 NOT – logical invert accu<br>9 LOAD – load operand to accu | (don't care) | <operand> |

**Example:**

Multiply accu by -5000
*Mnemonic:* CALC MUL, -5000

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $13 | $02 | $00 | $FF | $FF | $EC | $78 |

## 3.6.17 COMP (compare)

The specified number is compared to the value in the accumulator register. The result of the comparison can for example be used by the conditional jump (JC) instruction. This command is intended for use in standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. It does not make sense to use this command in direct mode.

**Internal function:** The specified value is compared to the internal *accumulator*, which holds the value of a preceding *get* or calculate instruction (see GAP/GGP/ CALC/CALCX). The internal arithmetic status flags are set according to the comparison result.

**Related commands:** JC (jump conditional), GAP, GGP, CALC, CALCX

**Mnemonic:** COMP <value>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 20 | (don't care) | (don't care) | <comparison value> |

**Example:**

Jump to the address given by the label when the position of motor is greater than or equal to 1000.

```
GAP 1, 2, 0     //get axis parameter, type: no. 1 (actual position), motor: 0, value: 0 (don't care)
COMP 1000       //compare actual value to 1000
JC GE, Label    //jump, type: 5 greater/equal, the label must be defined somewhere else in the
                program
```

*Binary format of the COMP 1000 command:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $14 | $00 | $00 | $00 | $00 | $03 | $e8 |

## 3.6.18 JC (jump conditional)

The JC instruction enables a conditional jump to a fixed address in the TMCL program memory, if the specified condition is met. The conditions refer to the result of a preceding comparison. Please refer to COMP instruction for examples. This function is for standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. It does not make sense to use this command in direct mode. See the host-only control functions for details.

**Internal function:** the TMCL program counter is set to the passed value if the arithmetic status flags are in the appropriate state(s).

**Related commands:** JA, COMP, WAIT, CLE

**Mnemonic:** JC <condition>, <label>
where <condition>=ZE|NZ|EQ|NE|GT|GE|LT|LE|ETO|EAL|EDV|EPO

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 21 | 0 ZE – zero<br>1 NZ – not zero<br>1 EQ – equal<br>1 NE – not equal<br>1 GT – greater<br>1 GE – greater/equal<br>6 LT – lower<br>7 LE – lower/equal<br>8 ETO – time out error | (don't care) | <jump address> |

**Example:**

Jump to address given by the label when the position of motor is greater than or equal to 1000.

```
GAP 1, 0, 0      //get axis parameter, type: no. 1 (actual position), motor: 0, value: 0 (don't care)
COMP 1000        //compare actual value to 1000
JC GE, Label     //jump, type: 5 greater/equal
...
...
Label: ROL 0, 1000
```

*Binary format of JC GE, Label when Label is at address 10:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $15 | $05 | $00 | $00 | $00 | $00 | $0a |

### 3.6.19 JA (jump always)

Jump to a fixed address in the TMCL program memory. This command is intended for standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. This command cannot be used in direct mode.

**Internal function:** the TMCL program counter is set to the passed value.

**Related commands:** JC, WAIT, CSUB

**Mnemonic:** JA <Label>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 22 | (don't care) | (don't care) | <jump address> |

**Example:** An infinite loop in TMCL™

```
Loop:  MVP ABS, 0, 10000
       WAIT POS, 0, 0
       MVP ABS, 0, 0
       WAIT POS, 0, 0
       JA Loop          //Jump to the label Loop
```

*Binary format of JA Loop assuming that the label Loop is at address 20:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $16 | $00 | $00 | $00 | $00 | $00 | $14 |

## 3.6.20 CSUB (call subroutine)

This function calls a subroutine in the TMCL program memory. It is intended for standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. This command cannot be used in direct mode.

**Internal function:** The actual TMCL program counter value is saved to an internal stack, afterwards overwritten with the passed value. The number of entries in the internal stack is limited to 8. This also limits nesting of subroutine calls to 8. The command will be ignored if there is no more stack space left.

**Related commands:** RSUB, JA

**Mnemonic:** CSUB <Label>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 23 | (don't care) | (don't care) | <subroutine address> |

**Example: Call a subroutine**

```
Loop:  MVP ABS, 0, 10000
       CSUB SubW      //Save program counter and jump to label SubW
       MVP ABS, 0, 0
       JA Loop

SubW:  WAIT POS, 0, 0
       WAIT TICKS, 0, 50
       RSUB           //Continue with the command following the CSUB command
```

*Binary format of the CSUB SubW command assuming that the label SubW is at address 100:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $17 | $00 | $00 | $00 | $00 | $00 | $64 |

## 3.6.21 RSUB (return from subroutine)

Return from a subroutine to the command after the CSUB command. This command is intended for use in standalone mode only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. This command cannot be used in direct mode.

**Internal function:** The TMCL program counter is set to the last value of the stack. The command will be ignored if the stack is empty.

**Related command:** CSUB

**Mnemonic:** RSUB

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 24 | (don't care) | (don't care) | (don't care) |

**Example:** please see the CSUB example (section 3.6.20).

*Binary format of RSUB:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $18 | $00 | $00 | $00 | $00 | $00 | $00 |

## 3.6.22 WAIT (wait for an event to occur)

This instruction interrupts the execution of the TMCL program until the specified condition is met. This command is intended for standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL program memory while the program loads down. This command cannot be used in direct mode.

**THERE ARE FIVE DIFFERENT WAIT CONDITIONS THAT CAN BE USED:**

- TICKS: Wait until the number of timer ticks specified by the <ticks> parameter has been reached.
- POS: Wait until the target position of the motor specified by the <motor> parameter has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- REFSW: Wait until the reference switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- LIMSW: Wait until a limit switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- RFS: Wait until the reference search of the motor specified by the <motor> field has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.

The timeout flag (ETO) will be set after a timeout limit has been reached. You can then use a JC ETO command to check for such errors or clear the error using the CLE command.

**Internal function:** The TMCL program counter is held until the specified condition is met.

**Related commands:** JC, CLE

**Mnemonic:** WAIT <condition>, 0, <ticks>
where <condition> is TICKS|POS|REFSW|LIMSW|RFS

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 27 | 0 TICKS – timer ticks*[1] | don't care | <no. of ticks*[1]> |
| | 1 POS – target position reached | 0*[2] | <no. of ticks*[1] for timeout>, 0 for no timeout |
| | 2 REFSW – reference switch | 0*[2] | <no. of ticks*[1] for timeout>, 0 for no timeout |
| | 3 LIMSW – limit switch | 0*[2] | <no. of ticks*[1] for timeout>, 0 for no timeout |
| | 4 RFS – reference search completed | 0*[2] | <no. of ticks*[1] for timeout>, 0 for no timeout |

***[1] one tick is 10 milliseconds (in standard firmware)**
***[2] motor number is always 0 as only one motor is involved**

**Example:**
Wait for motor to reach its target position, without timeout
*Mnemonic:* WAIT POS, 0, 0

*Binary*:

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $1b | $01 | $00 | $00 | $00 | $00 | $00 |

### 3.6.23 STOP (stop TMCL program execution)

This function stops executing a TMCL program. The host address and the reply are only used to transfer the instruction to the TMCL program memory.

This command should be placed at the end of every standalone TMCL program. It is not to be used in direct mode.

**Internal function:** TMCL instruction fetching is stopped.

**Related commands:** none
**Mnemonic:** STOP

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 28 | (don't care) | (don't care) | (don't care) |

**Example:**
*Mnemonic:* STOP

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $1c | $00 | $00 | $00 | $00 | $00 | $00 |

## 3.6.24 SCO (set coordinate)

Up to 20 position values (coordinates) can be stored for every axis for use with the MVP COORD command. This command sets a coordinate to a specified value. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Please note that the coordinate number 0 is always stored in RAM only.

**Internal function:** the passed value is stored in the internal position array.

**Related commands:** GCO, CCO, MVP

**Mnemonic:** SCO <coordinate number>, 0, <position>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 30 | <coordinate number><br>0... 20 | 0 | <position><br>$-2^{31}$... $+2^{31}$ |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Set coordinate #1 of motor to 1000
*Mnemonic:* SCO 1, 0, 1000

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $1e | $01 | $00 | $00 | $00 | $03 | $e8 |

***Note***:
Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate to the EEPROM.

**THESE FUNCTIONS CAN BE ACCESSED USING THE FOLLOWING SPECIAL FORMS OF THE SCO COMMAND:**

| | |
|---|---|
| SCO 0, 255, 0 | copies all coordinates (except coordinate number 0) from RAM to the EEPROM. |
| SCO <coordinate number>, 255, 0 | copies the coordinate selected by <coordinate number> to the EEPROM. The coordinate number must be a value between 1 and 20. |

## 3.6.25 GCO (get coordinate)

This command makes possible to read out a previously stored coordinate. In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditioned jumps. In direct mode, the value is only output in the value field of the reply, without affecting the accumulator. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM, only).

Please note that the coordinate number 0 is always stored in RAM, only.

**Internal function:** the desired value is read out of the internal coordinate array, copied to the accumulator register and – in direct mode – returned in the *value* field of the reply.

**Related commands:** SCO, CCO, MVP

**Mnemonic:** GCO <coordinate number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 31 | <coordinate number><br>0... 20 | 0 | don't care |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Get motor value of coordinate 1
*Mnemonic:* GCO 1, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $1f | $01 | $00 | $00 | $00 | $00 | $00 |

*Reply:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $02 | $01 | $64 | $0a | $00 | $00 | $00 | $00 |

⇨ **Value: 0**

***Note***:
Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate from the EEPROM to the RAM.

**THESE FUNCTIONS CAN BE ACCESSED USING THE FOLLOWING SPECIAL FORMS OF THE GCO COMMAND:**

| | |
|---|---|
| GCO 0, 255, 0 | copies all coordinates (except coordinate number 0) from the EEPROM to the RAM. |
| GCO <coordinate number>, 255, 0 | copies the coordinate selected by <coordinate number> from the EEPROM to the RAM. The coordinate number must be a value between 1 and 20. |

## 3.6.26 CCO (capture coordinate)

The actual position of the axis is copied to the selected coordinate variable. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only). Please see the SCO and GCO commands on how to copy coordinates between RAM and EEPROM.

Note, that the coordinate number 0 is always stored in RAM only.

**Internal function:** the selected (24 bit) position values are written to the 20 by 3 bytes wide coordinate array.

**Related commands:** SCO, GCO, MVP

**Mnemonic:** CCO <coordinate number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 32 | <coordinate number><br>0... 20 | 0 | don't care |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Store current position of the axis 0 to coordinate 3
*Mnemonic:* CCO 3, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $20 | $03 | $00 | $00 | $00 | $00 | $00 |

## 3.6.27 ACO (accu to coordinate)

With the ACO command the actual value of the accumulator is copied to a selected coordinate of the motor. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Please note also that the coordinate number 0 is always stored in RAM only. For Information about storing coordinates refer to the SCO command.

**Internal function:** the actual value of the accumulator is stored in the internal position array.

**Related commands:** GCO, CCO, MVP COORD, SCO

**Mnemonic:** ACO <coordinate number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 39 | <coordinate number><br>0... 20 | 0 | don't care |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Copy the actual value of the accumulator to coordinate 1 of motor 0
*Mnemonic:* ACO 1, 0

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $27 | $01 | $00 | $00 | $00 | $00 | $00 |

## 3.6.28 CALCX (calculate using the X register)

This instruction is very similar to CALC, but the second operand comes from the X register. The X register can be loaded with the LOAD or the SWAP type of this instruction. The result is written back to the accumulator for further processing like comparisons or data transfer.

**Related commands:** CALC, COMP, JC, AAP, AGP

**Mnemonic:** CALCX <operation>
with <operation>=ADD|SUB|MUL|DIV|MOD|AND|OR|XOR|NOT|LOAD|SWAP

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 33 | 0 ADD – add X register to accu<br>1 SUB – subtract X register from accu<br>1 MUL – multiply accu by X register<br>1 DIV – divide accu by X-register<br>1 MOD – modulo divide accu by x-register<br>1 AND – logical and accu with X-register<br>6 OR – logical or accu with X-register<br>7 XOR – logical exor accu with X-register<br>8 NOT – logical invert X-register<br>9 LOAD – load accu to X-register<br>10 SWAP – swap accu with X-register | (don't care) | (don't care) |

**Example:**
Multiply accu by X-register
*Mnemonic:* CALCX MUL
*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $21 | $02 | $00 | $00 | $00 | $00 | $00 |

## 3.6.29 AAP (accumulator to axis parameter)

The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction.

For a table with parameters and values which can be used together with this command please refer to chapter 4.

**Related commands:** AGP, SAP, GAP, SGP, GGP, CALC, CALCX

**Mnemonic:** AAP <parameter number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 34 | <parameter number> | 0* | <don't care> |

*** Motor number is always 0 as only one motor is involved**

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Positioning motor by a potentiometer connected to the analogue input #0:

```
Start:  GIO 0,1       // get value of analogue input line 0
        CALC MUL, 4   // multiply by 4
        AAP 0,0       // transfer result to target position of motor 0
        JA Start      // jump back to start
```

*Binary format of the AAP 0,0 command:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $22 | $00 | $00 | $00 | $00 | $00 | $00 |

## 3.6.30 AGP (accumulator to global parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction.

*Note*:
The global parameters in bank 0 are EEPROM-only and thus should not be modified automatically by a standalone application. (See chapter 5 for a complete list of global parameters).

**Related commands:** AAP, SGP, GGP, SAP, GAP

**Mnemonic:** AGP <parameter number>, <bank number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 35 | <parameter number> | <bank number> | (don't care) |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 5.

**Example:**

Copy accumulator to TMCL user variable #3
*Mnemonic:* AGP 3, 2

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $23 | $03 | $02 | $00 | $00 | $00 | $00 |

## 3.6.31 CLE (clear error flags)

This command clears the internal error flags. ***It is intended for use in standalone mode only and must not be used in direct mode.***

**The following error flags can be cleared by this command (determined by the <flag> parameter):**

- ALL: clear all error flags.
- ETO: clear the timeout flag.
- EDV: clear the deviation flag

**Related commands:** JC

**Mnemonic:** CLE <flags>
where <flags>=ALL|ETO|EDV|EPO

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 36 | 0 – (ALL) all flags<br>1 – (ETO) timeout flag<br>3 – (EDV) deviation flag | (don't care) | (don't care) |

**Example:**

Reset the timeout flag
*Mnemonic:* CLE ETO

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $24 | $01 | $00 | $00 | $00 | $00 | $00 |

## 3.6.32 VECT (set interrupt vector)

The VECT command defines an interrupt vector. It needs an interrupt number and a label as parameter (like in JA, JC and CSUB commands).

***This label must be the entry point of the interrupt handling routine.***

**Related commands:** EI, DI, RETI

**Mnemonic:** VECT <interrupt number>, <label>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 37 | <interrupt number> | don't care | <label> |

**THE FOLLOWING TABLE SHOWS ALL INTERRUPT VECTORS THAT CAN BE USED:**

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | (Target) position reached |
| 15 | Stall (stallGuard2) |
| 21 | Deviation |
| 27 | Stop left |
| 28 | Stop right |
| 39 | IN_0 change |
| 40 | IN_1 change |
| 41 | IN_2 change |
| 42 | IN_3 change |

**Example:** Define interrupt vector at target position 500
VECT 3, 500

*Binary format of VECT:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $25 | $03 | $00 | $00 | $00 | $01 | $F4 |

## 3.6.33 EI (enable interrupt)

The EI command enables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally enables interrupts.

**Related command:** DI, VECT, RETI

**Mnemonic:** EI <interrupt number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 25 | <interrupt number> | don't care | don't care |

**THE FOLLOWING TABLE SHOWS ALL INTERRUPT VECTORS THAT CAN BE USED:**

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | (Target) position reached |
| 15 | Stall (stallGuard2™) |
| 21 | Deviation |
| 27 | Stop left |
| 28 | Stop right |
| 39 | IN_0 change |
| 40 | IN_1 change |
| 41 | IN_2 change |
| 42 | IN_3 change |

**Examples:**

Enable interrupts globally
EI, 255

*Binary format of EI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $19 | $FF | $00 | $00 | $00 | $00 | $00 |

Enable interrupt when target position reached
EI, 3

*Binary format of EI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $19 | $03 | $00 | $00 | $00 | $00 | $00 |

## 3.6.34 DI (disable interrupt)

The DI command disables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally disables interrupts.

**Related command:** EI, VECT, RETI

**Mnemonic:** DI <interrupt number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 26 | <interrupt number> | don't care | don't care |

**THE FOLLOWING TABLE SHOWS ALL INTERRUPT VECTORS THAT CAN BE USED:**

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | (Target) position reached |
| 15 | Stall (stallGuard2™) |
| 21 | Deviation |
| 27 | Stop left |
| 28 | Stop right |
| 39 | IN_0 change |
| 40 | IN_1 change |
| 41 | IN_2 change |
| 42 | IN_3 change |

**Examples:**

Disable interrupts globally
DI, 255

*Binary format of DI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $1A | $FF | $00 | $00 | $00 | $00 | $00 |

Disable interrupt when target position reached
DI, 3

*Binary format of DI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| Value (hex) | $01 | $1A | $03 | $00 | $00 | $00 | $00 | $00 |

### 3.6.35 RETI (return from interrupt)

This command terminates the interrupt handling routine, and the normal program execution continues.

*At the end of an interrupt handling routine the RETI command must be executed.*

**Internal function:** the saved registers (A register, X register, flags) are copied back. Normal program execution continues.

**Related commands:** EI, DI, VECT

**Mnemonic:** RETI

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 38 | don't care | don't care | don't care |

**Example:** Terminate interrupt handling and continue with normal program execution
RETI

*Binary format of RETI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $01 | $26 | $00 | $00 | $00 | $00 | $01 | $00 |

### 3.6.36 Customer Specific TMCL Command Extension (UF0... UF7 / User Function)

The user definable functions UF0... UF7 are predefined, functions without topic for user specific purposes. Contact TRINAMIC for the customer specific programming of these functions.

**Internal function:** Call user specific functions implemented in *C* by TRINAMIC.

**Related commands:** none

**Mnemonic:** UF0... UF7

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 64... 71 | (user defined) | (user defined) | (user defined) |

**Reply in direct mode:**

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $02 | $01 | (user defined) | 64... 71 | (user defined) | (user defined) | (user defined) | (user defined) |

### 3.6.37 Request Target Position Reached Event

This command is the only exception to the TMCL protocol, as it sends two replies: One immediately after the command has been executed (like all other commands also), and one additional reply that will be sent when the motor has reached its target position.

This instruction can only be used in direct mode (in standalone mode, it is covered by the WAIT command) and hence does not have a mnemonic.

**Internal function:** Send an additional reply when the motor has reached its target position

**Mnemonic: ---**

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 138 | 0/1 | (don't care) | 1 |

**Reply in direct mode (right after execution of this command):**

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $02 | $01 | 100 | 138 | $00 | $00 | $00 | Motor bit mask |

**Additional reply in direct mode (after motors have reached their target positions):**

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 |
| **Value (hex)** | $02 | $01 | 128 | 138 | $00 | $00 | $00 | Motor bit mask |

## 3.6.38 TMCL Control Functions

The following functions are for host control purposes only and are not allowed for standalone mode.
In most cases, there is no need for the customer to use one of those functions (except command 139).

TMCL control commands have no mnemonics, as they cannot be used in TMCL programs. These Functions are to be used only by the TMCL-IDE (e.g. to download a TMCL application into the module).

**CONTROL COMMANDS THAT COULD BE USEFUL FOR A USER HOST APPLICATION ARE:**

- *get firmware revision* (command 136, please note the special reply format of this command, described at the end of this section)
- *run application* (command 129)

*All other functions can be achieved by using the appropriate functions of the TMCL-IDE!*

| Instruction | Description | Type | Mot/Bank | Value |
|---|---|---|---|---|
| 128 – stop application | a running TMCL standalone application is stopped | (don't care) | (don't care) | (don't care) |
| 129 – run application | TMCL execution is started (or continued) | 0 – run from current address<br>1 – run from specified address | (don't care) | (don't care)<br><br>starting address |
| 130 – step application | only the next command of a TMCL application is executed | (don't care) | (don't care) | (don't care) |
| 131 – reset application | the program counter is set to zero, and the standalone application is stopped (when running or stepped) | (don't care) | (don't care) | (don't care) |
| 132 – start download mode | target command execution is stopped and all following commands are transferred to the TMCL memory | (don't care) | (don't care) | starting address of the application |
| 133 – quit download mode | target command execution is resumed | (don't care) | (don't care) | (don't care) |
| 134 – read TMCL memory | the specified program memory location is read | (don't care) | (don't care) | <memory address> |
| 135 – get application status | one of these values is returned:<br>0 – stop<br>1 – run<br>1 – step<br>3 – reset | (don't care) | (don't care) | (don't care) |
| 136 – get firmware version | return the module type and firmware revision either as a string or in binary format | 0 – string<br>1 – binary | (don't care) | (don't care) |
| 137 – restore factory settings | reset all settings stored in the EEPROM to their factory defaults<br>This command does not send back a reply. | (don't care) | (don't care) | must be 1234 |

**SPECIAL REPLY FORMAT OF COMMAND 136:**

**Type set to 0 – reply as a string:**

| Byte index | Contents |
|---|---|
| 1 | Host Address |
| 2… 9 | Version string (8 characters, e.g. 1140V1.17) |

- There is no checksum in this reply format!
- To get also the last byte when using the CAN bus interface, just send this command in an eight byte frame instead of a seven byte frame. Then, eight bytes will be sent back, so you will get all characters of the version string.

**Type set to 1 – version number in binary format:**

- Please use the normal reply format.
- The version number is output in the *value* field of the reply in the following way:

| Byte index in value field | Contents |
|---|---|
| 1 | Version number, low byte |
| 2 | Version number, high byte |
| 3 | Type number, low byte |
| 4 | Type number, high byte |

# 4 Axis Parameters

The following sections describe all axis parameters that can be used with the SAP, GAP, AAP, STAP and RSAP commands.

**MEANING OF THE LETTERS IN COLUMN *ACCESS*:**

| Access type | Related command(s) | Description |
|---|---|---|
| R | GAP | Parameter readable |
| W | SAP, AAP | Parameter writable |
| E | STAP, RSAP | Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STAP command and also explicitly restored (copied back from EEPROM into RAM) using RSAP. |

*Basic parameters should be adjusted to motor / application for proper module operation.*

*Parameters for the more experienced user – please do not change unless you are absolutely sure.*

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 0 | Target (next) position | The desired position in position mode (see ramp mode, no. 138). | ± $2^{31}$-1 [µsteps] | RW |
| 1 | Actual position | The current position of the motor. Should only be overwritten for reference point setting. | ± $2^{31}$-1 [µsteps] | RW |
| 2 | Target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 3 | Actual speed | The current rotation speed. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 4 | Maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RWE |
| 5 | Maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 429 datasheet for calculation of physical units. | 0… 2047* | RWE |

<table>
<tr><th>Number</th><th>Axis Parameter</th><th>Description</th><th>Range [Unit]</th><th>Acc.</th></tr>
<tr><td>6</td><td>Absolute max. current<br>(CS / Current Scale)</td><td>The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps.
<table>
<tr><td>0… 7</td><td>79…87</td><td>160… 167</td><td>240… 247</td></tr>
<tr><td>8… 15</td><td>88… 95</td><td>168… 175</td><td>248… 255</td></tr>
<tr><td>16… 23</td><td>96… 103</td><td>176… 183</td><td></td></tr>
<tr><td>24… 31</td><td>104… 111</td><td>184… 191</td><td></td></tr>
<tr><td>32… 39</td><td>112… 119</td><td>192… 199</td><td></td></tr>
<tr><td>40… 47</td><td>120… 127</td><td>200… 207</td><td></td></tr>
<tr><td>48… 55</td><td>128… 135</td><td>208… 215</td><td></td></tr>
<tr><td>56… 63</td><td>136… 143</td><td>216… 223</td><td></td></tr>
<tr><td>64… 71</td><td>144… 151</td><td>224… 231</td><td></td></tr>
<tr><td>72… 79</td><td>152… 159</td><td>232… 239</td><td></td></tr>
</table>
The most important motor setting, since too high values might cause motor damage!</td><td>0… 255<br>$I_{peak} =< value >\times \frac{2.8A}{255}$<br>$I_{RMS} =< value >\times \frac{2A}{255}$</td><td>RWE</td></tr>
<tr><td>7</td><td>Standby current</td><td>The current limit two seconds after the motor has stopped.</td><td>0… 255<br>$I_{peak} =< value >\times \frac{2.8A}{255}$<br>$I_{RMS} =< value >\times \frac{2A}{255}$</td><td>RWE</td></tr>
<tr><td>8</td><td>Target pos. reached</td><td>Indicates that the actual position equals the target position.</td><td>0/1</td><td>R</td></tr>
<tr><td>9</td><td>Ref. switch status</td><td>The logical state of the reference home switch.</td><td>0/1</td><td>R</td></tr>
<tr><td>10</td><td>Right limit switch status</td><td>The logical state of the (right) limit switch.</td><td>0/1</td><td>R</td></tr>
<tr><td>11</td><td>Left limit switch status</td><td>The logical state of the left limit switch (in three switch mode)</td><td>0/1</td><td>R</td></tr>
<tr><td>12</td><td>Right limit switch disable</td><td>If set, deactivates the stop function of the right switch</td><td>0/1</td><td>RWE</td></tr>
<tr><td>13</td><td>Left limit switch disable</td><td>Deactivates the stop function of the left switch resp. reference switch if set.</td><td>0/1</td><td>RWE</td></tr>
<tr><td>130</td><td>Minimum speed</td><td>Should always be set 1 to ensure exact reaching of the target position.</td><td>0… 2047<br>Default = 1<br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$</td><td>RWE</td></tr>
<tr><td>135</td><td>Actual acceleration</td><td>The current acceleration (read only).</td><td>0… 2047*</td><td>R</td></tr>
<tr><td>138</td><td>Ramp mode</td><td>Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected.</td><td>0/1/2</td><td>RWE</td></tr>
</table>

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 140 | Microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | 0… 8 | RWE |
| 149 | Soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. | 0/1 | RWE |
| 153 | Ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). | 0… 13 | RWE |
| 154 | Pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). | 0… 13 | RWE |
| 160 | Step interpolation enable | Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved.<br>0 – step interpolation off<br>1 – step interpolation on | 0/1 | RW |
| 162 | Chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good. | 0… 3 | RW |
| 163 | Chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>1 – classic const. off time | 0/1 | RW |
| 164 | Chopper hysteresis decrement | Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time.<br>0 – fast decrement<br>3 – very slow decrement | 0… 3 | RW |
| 165 | Chopper hysteresis end | Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.<br>-3… -1 negative hysteresis end setting<br>0 zero hysteresis end setting<br>1… 12 positive hysteresis end setting | -3… 12 | RW |
| 166 | Chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. | 0… 8 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 167 | Chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br><br>Off time setting for constant $t_{Off}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br><br>Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel. | 0 / 2... 15 | RW |
| 168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>Minimum motor current:<br>0 – ½ of CS<br>1 – ¼ of CS | 0/1 | RW |
| 169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0... 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement | 0... 3 | RW |
| 170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32<br>Upper stallGuard threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 | 0... 15 | RW |
| 171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>Current increment step size:<br>Scaling: 0... 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load | 1... 3 | RW |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). | 0... 15 | RW |
| 173 | stallGuard2 filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode | 0/1 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 174 | stallGuard2 threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0 \| Indifferent value<br>1… 63 \| less sensitivity<br>-1… -64 \| higher sensitivity | -64… 63 | RW |
| 175 | Slope control high side | Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope | 0… 3 | RW |
| 176 | Slope control low side | Determines the slope of the motor driver outputs. *Set identical to slope control high side.* | 0… 3 | RW |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* | 0/1 | RW |
| 178 | Short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* | 0..3 | RW |
| 179 | Vsense | sense resistor voltage based current scaling<br>0: Full scale sense resistor voltage is 1/18 VDD<br>1: Full scale sense resistor voltage is 1/36 VDD (refers to a current setting of 31 and DAC value 255)<br>*Use default value. Do not change!* | 0/1 | RW |
| 180 | smartEnergy actual current | This status value provides the *actual motor current* setting as controlled by coolStep™. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN.<br>Actual motor current scaling factor:<br>0 … 31: 1/32, 2/32, … 32/32 | 0… 31 | RW |
| 181 | Stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. | 0… 2047 | RW |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{pulse\_divisor} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. | 0… 255<br>$I_{peak} =< value >\times \frac{2.8A}{255}$<br>$I_{RMS} =< value >\times \frac{2A}{255}$ | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 193 | Ref. search mode | 1 search left stop switch only<br>2 search right stop switch, then search left stop switch<br>3 search right stop switch, then search left stop switch from both sides<br>4 search left stop switch from both sides<br>5 search home switch in negative direction, reverse the direction when left stop switch reached<br>6 search home switch in positive direction, reverse the direction when right stop switch reached<br>7 search home switch in negative direction, ignore end switches<br>8 search home switch in positive direction, ignore end switches<br><br>Additional functions:<br>- *Add 128 to a mode value for inverting the home switch (can be used with mode 5… 8).*<br>- *Add 64 to a mode for driving the right instead of the left reference switch (can be used with mode 1… 4).* | 1… 8 | RWE |
| 194 | Referencing search speed | For the reference search this value directly specifies the search speed. | 0… 2047 | RWE |
| 195 | Referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. | 0… 2047 | RWE |
| 196 | Distance end switches | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). | 0… 2.147.483.647 | R |
| 197 | Last reference position | Reference search: the last position before setting the counter to zero can be read out. | $-2^{31}$… $2^{31}-1$<br>[µsteps] | R |
| 200 | Boost current | Current used for acceleration and deceleration phases.<br>If set to 0 the same current as set by axis parameter 6 will be used. | 0… 255<br>$I_{peak} = < value > \times \frac{2.8A}{255}$<br>$I_{RMS} = < value > \times \frac{2A}{255}$ | RWE |
| 204 | Freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. | 0… 65535<br>0 = never<br>[msec] | RWE |
| 206 | Actual load value | Readout of the actual load value with used for stall detection (stallGuard2™). | 0… 1023 | R |
| 207 | Extended error flags | 1 Motor stopped because of stallGuard2 detection.<br>2 Motor stopped because of encoder deviation.<br>3 Motor stopped because of (1) and (2).<br>*Will be reset automatically by the next motion command.* | 1… 3 | R |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 208 | TMC262 driver error flags | Bit 0: stallGuard2 status (1: threshold reached)<br>Bit 1: Overtemperature (1: driver is shut down due to overtemperature)<br>Bit 2: Pre-warning overtemperature (1: Threshold is exceeded)<br>Bit 3: Short to ground A (1: Short condition detected, driver currently shut down)<br>Bit 4: Short to ground B (1: Short condition detected, driver currently shut down)<br>Bit 5: Open load A (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 6: Open load B (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 7: Stand still (1: No step impulse occurred on the step input during the last 2^20 clock cycles) | 0/1 | R |
| 209 | Encoder position *sensOstep* | The value of an encoder register can be read out or written. | [encoder steps] | RW |
| 210 | Encoder prescaler *sensOstep* | Prescaler for the sensOstep encoder. | See paragraph **Error! Reference source not found.** | RWE |
| 212 | Maximum encoder deviation *sensOstep* | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br>[encoder steps] | RWE |
| 214 | Power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). | 1… 65535<br>[10msec] | RWE |
| 215 | Absolute resolver value sensOstep | Absolute position of the internal sensOstep encoder. The absolute position is within one motor rotation. | 0 .. 1023 | R |
| 216 | Encoder position *external encoder* | The value of the external encoder register can be read out or written.<br>*This parameter is only used if an external encoder is connected.* | [encoder steps] | RW |
| 217 | Encoder prescaler *external encoder* | Prescaler for external encoder. | See paragraph **Error! Reference source not found.** | |
| 218 | Maximum encoder deviation *external encoder* | When the actual position (parameter 1) and the encoder position (parameter 216) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero.<br>*This parameter is only used if an external encoder is connected.* | 0… 65535<br>[encoder steps] | |

***** Unit of acceleration: $\frac{16MHz^2}{536870912 \cdot 2^{puls\_divisor+ramp\_divisor}} \frac{\text{microsteps}}{\text{sec}^2}$

## 4.1 stallGuard2

The module is equipped with TMC262 motor driver chip. The TMC262 features load measurement that can be used for stall detection. stallGuard2 delivers a sensorless load measurement of the motor as well as a stall detection signal. The measured value changes linear with the load on the motor in a wide range of load, velocity and current settings. At maximum motor load the stallGuard2 value goes to zero. This corresponds to a load angle of 90° between the magnetic field of the stator and magnets in the rotor. This also is the most energy efficient point of operation for the motor.

Stall detection means that the motor will be stopped when the load gets too high. It is configured by axis parameter #174.

*Stall detection can also be used for finding the reference point. Do not use RFS in this case.*

## 4.2 coolStep Related Axis Parameters

The figure below gives an overview of the coolStep related parameters. Please have in mind that the figure shows only one example for a drive. There are parameters which concern the configuration of the current. Other parameters are for velocity regulation and for time adjustment.

**THE FOLLOWING ADJUSTMENTS HAVE TO BE MADE:**

- Thresholds for current ($I_6$, $I_7$ and $I_{183}$) and velocity ($V_{182}$) have to be identified and set.
- The stallGuard2 feature has to be adjusted and enabled with parameters $SG_{170}$ and $SG_{181}$.
- The reduction or increasing of the current in the coolStep area (depending on the load) has to be configured with parameters $I_{169}$ and $I_{171}$.

*In this chapter only basic axis parameters are mentioned which concern coolStep and stallGuard2. The complete list of axis parameters in chapter 4 contains further parameters which offer more configuration possibilities.*

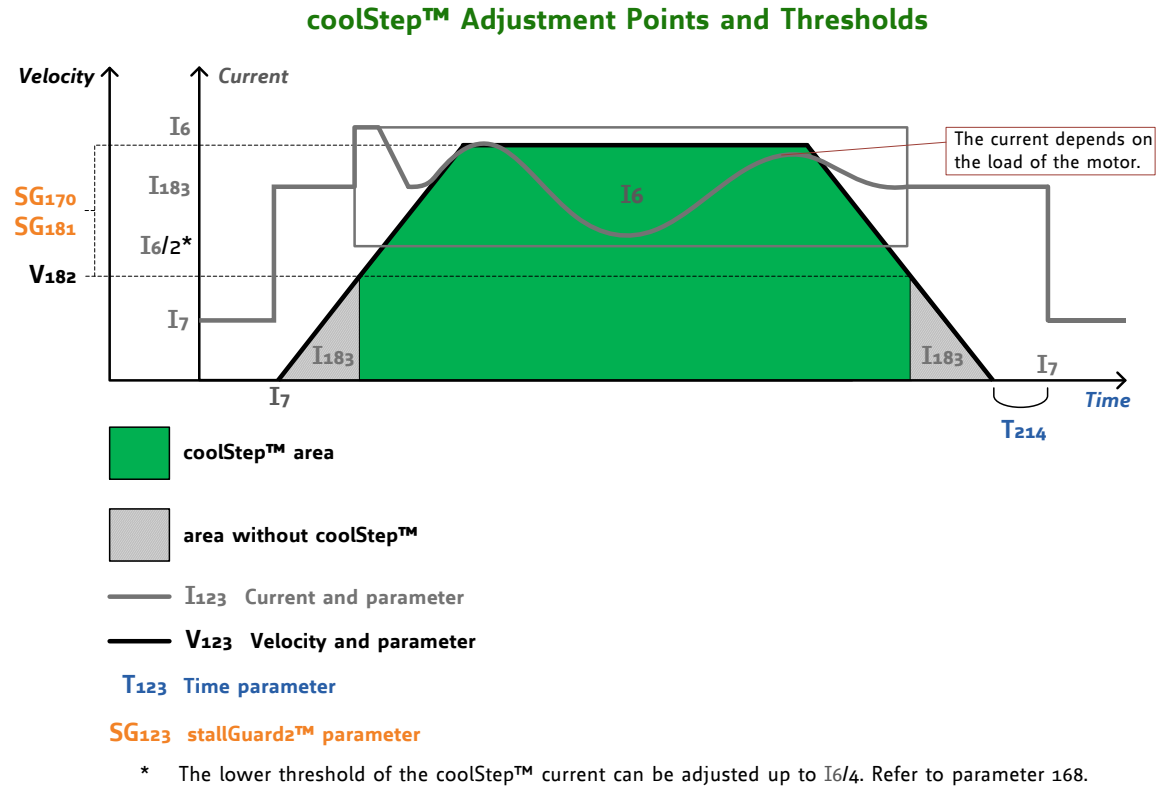



**Figure 4.1: coolStep™ adjustment points and thresholds**

### coolStep related Axis Parameters

*smartEnergy is an earlier name for coolStep.*

| Number | Axis parameter | Description |
|---|---|---|
| I6 | Absolute max. current (CS / Current Scale) | The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0... 255 and can be adjusted in 32 steps (0... 255 divided by eight; e.g. step 0 = 0... 7, step 1 = 8... 15 and so on).<br>*The most important motor setting, since too high values might cause motor damage!* |
| I7 | Standby current | The current limit two seconds after the motor has stopped. |
| I168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>Minimum motor current:<br>0 – ½ of CS<br>1 – ¼ of CS |
| I169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current. Number of stallGuard2™ measurements per decrement:<br>Scaling: 0... 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement |
| I171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>Current increment step size:<br>Scaling: 0... 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load |
| I183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. Please adjust the threshold speed with axis parameter 182. |
| SG170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased. |
| SG181 | Stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. |
| V182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. |
| T214 | Power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). |

*For further information about the coolStep™ feature please refer to the TMC262 Datasheet.*

## 4.3 Reference Search

The built-in reference search feature (instruction RFS, no. 13) offers switching point calibration and supports up-to two end / stop / limit switches and one home switch. Depending on the selected reference search mode (see table below), one of the limit switches or the home switch may be used as reference switch and the end / limit switches are either respected or ignored during the reference search. The internal operation is based on a state machine that can be started, stopped and monitored at any time during the reference search operation (instruction RFS, no. 13).

**REMARKS ON REFERENCE SEARCH OPERATION:**

- End / limit switches: enable / disable of the automatic stop function corresponding to the end / limit switches (axis parameters 12 and 13) has no influence on the reference search.
- Initial search speed: until the reference switch is found for the first time, the reference search speed specified by axis parameter 194 is used. (see figures below for more details)
- Search speed after reference switch has been triggered: As soon as the reference switch has been detected the motor will switch to the reference switch speed (axis parameter 195). (see figures below for more details)
- Search one of the end / stop / limit switches (from one side): when hitting the reference switch during movement, the motor stops and slowly moves backwards until the switch is released again. Motor stops again, reverses direction and slowly moves forward until the reference switch is triggered again. The reference point will be set to the center between release and trigger point of the reference switch on one side. With this mode the reference switch is not "overrun" – instead the middle between the two transitions of the switch on one side open -> close and close -> open is used as reference.
- Search one of the end / stop / limit switches or the home switch from both sides: when hitting the reference switch during movement, the motor will continue moving until the switch is released again. Motor stops then, reverses direction and slowly moves forward until the switch is triggered again. The motor continues movement at low speed until the switch is released again, stops, reverses direction, moves towards the reference switch at low speed again until it is triggered again. It will then move to the middle of the switch between the two positions where the switch has been activated / triggered at low speed from both sides. With this mode the reference switch is "overrun" and the motor stops in the middle between activation of the switch from the left and the right side. This mode is usually used with the home switch.
- Using axis parameter 193 the reference search mode can be selected (see next chapter for more details on different reference search modes) before the actual reference search is started using the RFS command (command no. 13).
- End / limit / stop / Home switches: With default configuration it is expected that both end / limit / stop switches and the home switch are normally closed (N.C.) switches. These switches should be connected between the STOP_L (IN_1, Pin 6 of the multipurpose I/O connector), STOP_R (IN_2, Pin 7 of the multipurpose I/O connector) resp. HOME (IN_3, Pin 8 of the multipurpose I/O connector) input of and ground (GND). With default configuration the inputs offer pull-up resistors to +5V. As soon as one of the switches is activated (open) the input level will go high due to the internal pull-ups. For other types of switches and / or other external connections this can be changed in software. Internal pull-ups can be deactivated using the command SIO 0, 0, 0 (for all pull-ups at once – please see SIO command for more details and different configuration options). This way a normally open (N.O.) switch may be connected between supply voltage (e.g. +24V) and the STOP_L, STOP_R or HOME input. The input will be low as long as the switch is not activated and turning high as soon as the switch is activated. For other configurations it is also possible to invert the polarity of the input (see global parameter 79).

**AXIS PARAMETERS RELATED TO REFERENCE SEARCH**

| Number | Axis Parameter | Description |
|---|---|---|
| 9 | Reference switch status | The logical state of the HOME switch. |
| 10 | Right limit switch status | The logical state of the STOP_R switch. |
| 11 | Left limit switch status | The logical state of the STOP_L switch |
| 12 | Right limit switch disable | If set, deactivates the stop function of the STOP_R switch (not applicable for the reference search algorithm) |
| 13 | Left limit switch disable | If set, deactivates the stop function of the STOP_L switch (not applicable for the reference search algorithm) |
| 149 | Soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the end / stop / limit switch is hit. |
| 193 | Reference search mode | 1 search left stop switch STOP_L only<br>2 search right stop switch STOP_R, then search left stop switch STOP_L<br>3 search right stop switch STOP_R, then search left stop switch STOP_L from both sides<br>4 search left stop switch STOP_L from both sides<br>5 search home switch HOME in negative direction, reverse the direction when left stop switch STOP_L reached<br>6 search home switch HOME in positive direction, reverse the direction when right stop switch STOP_R reached<br>7 search home switch HOME in negative direction, ignore end switches STOP_L and STOP_R<br>8 search home switch HOME in positive direction, ignore end switches STOP_L and STOP_R<br>Additional functions:<br>- *Add 128 to a mode value for inverting the home switch HOME polarity (can be used with mode 5… 8).*<br>- *Add 64 to a mode for exchanging left (STOP_L) and right (STOP_R) stop switches (can be used with mode 1… 4).* |
| 194 | Reference search speed | For the reference search this value directly specifies the search speed. |
| 195 | Reference switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. |
| 196 | Reference switch distance | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). |

## 4.3.1 Reference Search Modes (Axis Parameter 193)

The following figures explain the actual motor movement during the different reference search modes in more detail. A linear stage with two end points and a moving slider is taken as example.

***SAP 193, 0, 1***

*Search left stop switch, only*

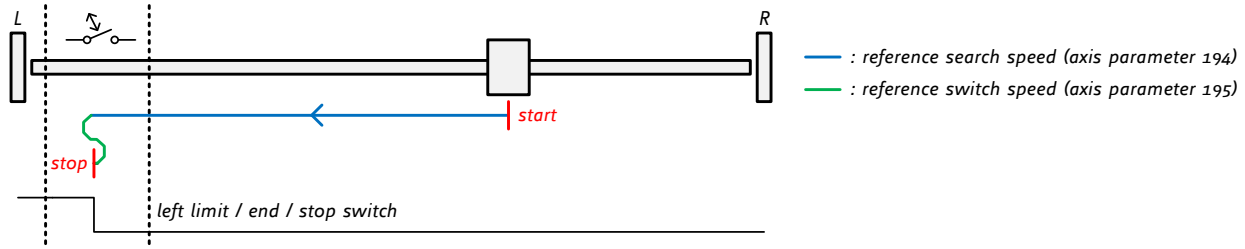


**Figure 5.2: Search left stop switch, only**

### *SAP 193, 0, 2*

*Search right stop switch, then search left stop switch*

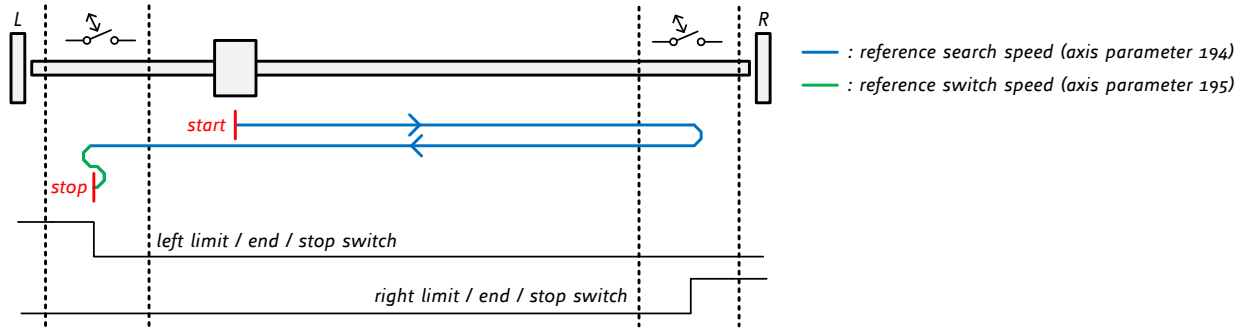

L
R
start
stop
left limit / end / stop switch
right limit / end / stop switch
: reference search speed (axis parameter 194)
: reference switch speed (axis parameter 195)

**Figure 5.3: Search right stop switch, then search left stop switch**

### *SAP 193, 0, 3*

*Search right stop switch, then search left stop switch from both sides*

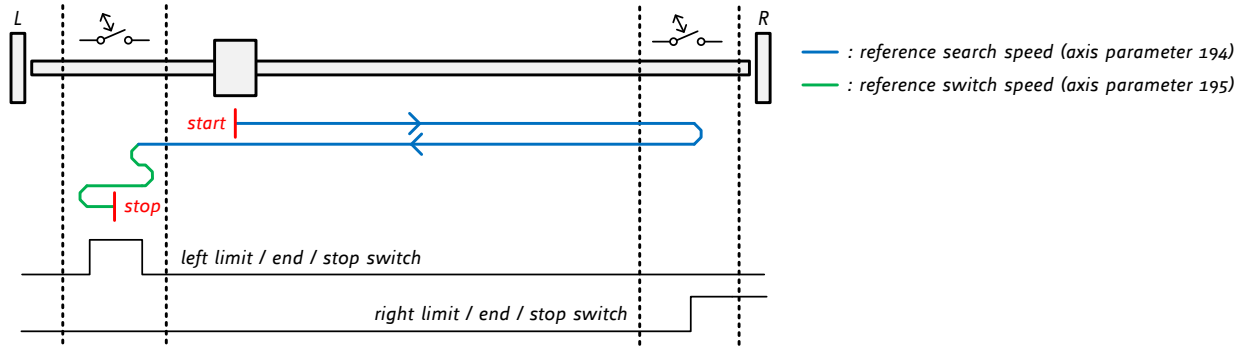

L
R
start
stop
left limit / end / stop switch
right limit / end / stop switch
: reference search speed (axis parameter 194)
: reference switch speed (axis parameter 195)

**Figure 5.4: Search right stop switch, then search left stop switch from both sides**

### *SAP 193, 0, 4*

*Search left stop switch from both sides*

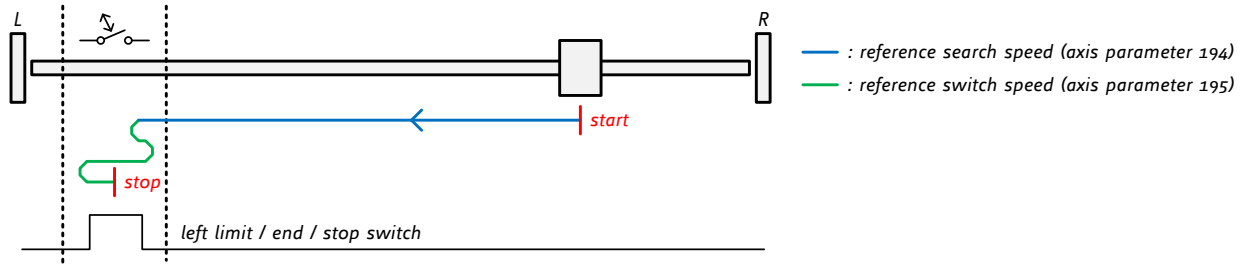



**Figure 5.5: Search left stop switch from both sides**

### *SAP 193, 0, 5*

*Search home switch in negative direction, reverse direction in case left stop switch is hit*

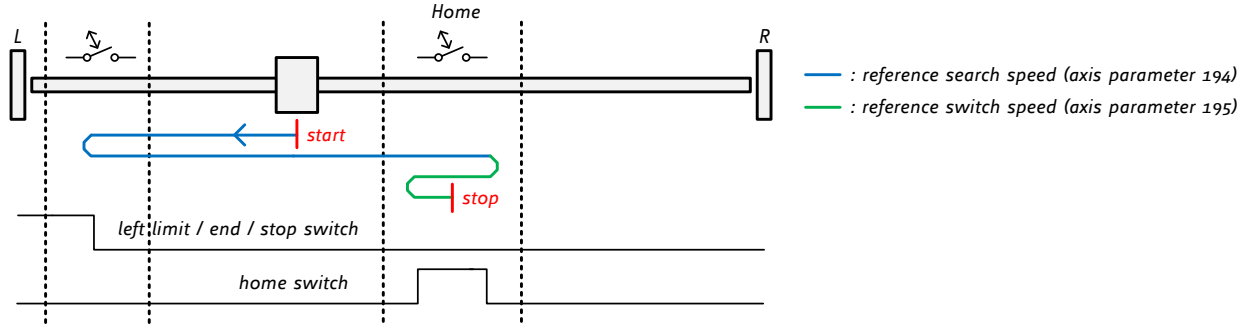



**Figure 5.6: Search home switch in negative direction, reverse direction in case left stop switch is hit**

### *SAP 193, 0, 6*

*Search home switch in positive direction, reverse direction in case right stop switch is hit*

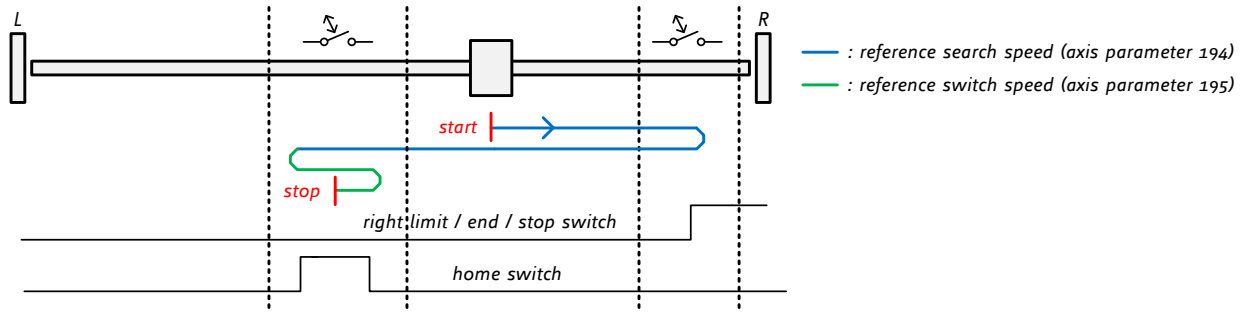



**Figure 5.7: Search home switch in positive direction, reverse direction in case right stop switch is hit**

### *SAP 193, 0, 7*

*Search home switch in negative direction, ignore end switches*

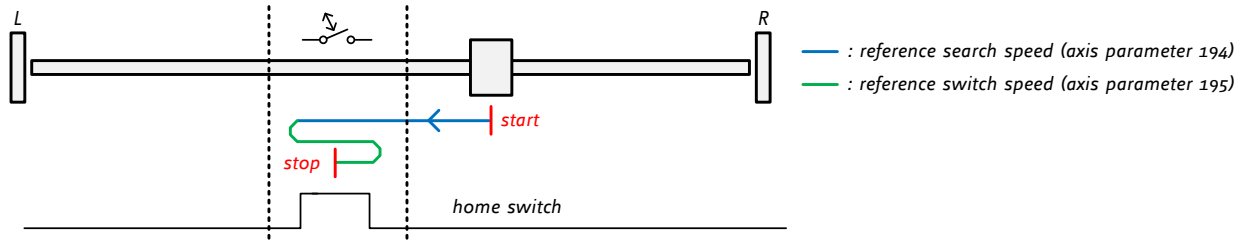



**Figure 5.8: Search home switch in negative direction, ignore end switches**

### *SAP 193, 0, 8*

*Search home switch in positive direction, ignore end switches*

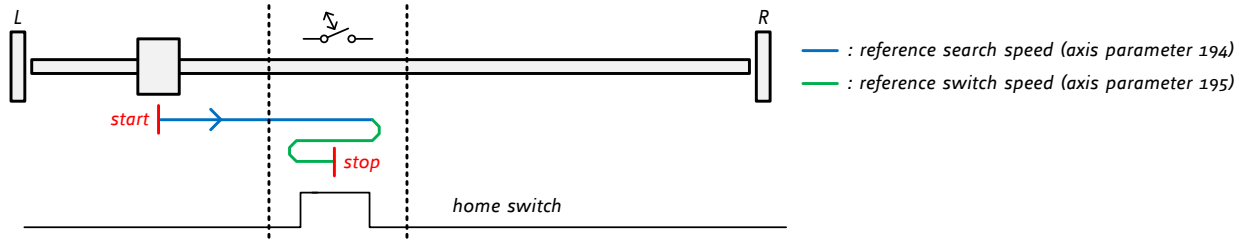



Figure 5.9: Search home switch in positive direction, ignore end switches

# 4.4 Changing the Prescaler Value of an Encoder

The TMCM-1140 module offers an integrated sensOstep encoder. The built-in encoder has 1024 steps/rotation. In addition, an external incremental a/b/n encoder might be connected to the Multipurpose I/O connector. This encoder can be used as an alternative to the internal one or in addition (e.g. to supervise motion at a different location than the motor axis e.g. at the other side of a belt / gearbox etc.).

*Note*:
The following examples about selecting a prescaler value will be valid for the internal sensOstep encoder and for external encoders, if their resolution is 1024 steps/rotation and a stepper motor with 200 fullsteps per rotation is used, only. For different encoder resolutions / different number of fullsteps per rotation new values have to be calculated.

**FOR THE OPERATION WITH ENCODER PLEASE CONSIDER THE FOLLOWING HINTS:**

- The encoder counter can be read out in software and can be used to control the exact position of the motor. This also makes basic closed loop operation possible.
- To read out or to change the position value of the internal encoder, axis parameter #209 is used (axis parameter #216 for the external encoder).
- So, to read out the position of your encoder *0* use *GAP 209, 0 (resp. GAP 216, 0)*. The position values can also be changed using command SAP 209, 0, *<new_position>*
- To change the encoder settings, axis parameter #210 is used (axis parameter #217 for external encoder). For changing the prescaler of the internal encoder use *SAP 210, 0, <p> (resp. SAP 217, 0, <p> for the external one)*.
- Automatic motor stop on deviation error is also available (e.g. for immediate step loss detection during motor movement). This can be set using axis parameter 212 (maximum deviation) resp. 218 for the external encoder. This function is turned off when the maximum deviation is set to 0.

Using the prescaler the encoder counter increments / decrements can be aligned to the position (microstep) counter. This is essential when using the deviation error supervision (parameter 212 resp. 218) and simplifies encoder counter vs. position/microstep counter comparisons (1:1).

*<number_of_microsteps_per_motor_rotation> = <prescaler> x <encoder_ticks_per_rotation>*

- *<number_of_microsteps_per_motor_rotation>*
  with default setting of 256 micrsteps per fullstep and a stepper motor with 200 fullsteps per rotation this is 256 x 200 = 51200 microsteps per motor rotation
- *<encoder_ticks_per_rotation>*
  1024 with the integrated sensOstep encoder. With incremental a/b/n encoders this is 4 x number of encoder lines.

The default settings result in a prescaler value of 51200 / 1024 = 50 (default, see table below).

The integer value <p> used with axis parameter 210 (internal encoder) resp. 217 (external encoder) can be calculated from the prescaler value using the following formula:

*<p> = <prescaler> x 512*

Example: *<prescaler>* = 50
50 x 512 = 25600 (p)

The table below shows a subset of those prescalers that can be selected. Also, other values between those given in the table can be used (e.g. a value for <p> of 512 results in no scaling at all (prescaler = 1) which is sometimes helpful in case the encoder resolution is unknown / not known for sure). Only the values 1, 2, 4, and 16 must not be used for <p> (because they are needed to select the special encoder function below or rather are reserved for intern usage).

**TO SELECT A PRESCALER FOR DIFFERENT MICROSTEP RESOLUTIONS THE FOLLOWING VALUES CAN BE USED FOR <P>:**

| Microstep resolution (axis parameter 140) | Resulting prescaler | Value for <p> | SAP 210, 0, <p> (use SAP 217, 0, <p> for the external encoder) |
|---|---|---|---|
| 8 (256 micro steps) | 50 *(default)* | 25600 | SAP 210, 0, 25600 |
| 7 (128 micro steps) | 25 | 12800 | SAP 210, 0, 12800 |
| 6 (64 micro steps) | 12.5 | 6400 | SAP 210, 0, 6400 |
| 5 (32 micro steps) | 6.25 | 3200 | SAP 210, 0, 3200 |
| 4 (16 micro steps) | 3.125 | 1600 | SAP 210, 0, 1600 |
| 3 (8 micro steps) | 1.5625 | 800 | SAP 210, 0, 800 |
| 2 (4 micro steps) | 0.78125 | 400 | SAP 210, 0, 400 |
| 1 (2 micro steps) | 0.390625 | 200 | SAP 210, 0, 200 |

### CLEAR ENCODER

There is one special function that can also be configured using <p>. For clearing the encoder add the following value to <p>.

| Adder for <p> | SAP 210, 0, <p> (use SAP 217, 0, <p> for the external encoder) |
|---|---|
| 4 | Clear encoder with next null channel event |

*Add up both <p> values from these tables to get the required value for the SAP 210 command.*

# 5 Global Parameters

**GLOBAL PARAMETERS ARE GROUPED INTO 4 BANKS:**

- bank 0 (global configuration of the module)
- bank 1 (user C variables)
- bank 2 (user TMCL variables)
- bank 3 (interrupt configuration)

Please use SGP and GGP commands to write and read global parameters.

## 5.1 Bank 0

Parameters with numbers from 64 on configure stuff like the serial address of the module RS485 baud rate or the CAN bit rate. Change these parameters to meet your needs. The best and easiest way to do this is to use the appropriate functions of the TMCL-IDE. The parameters with numbers between 64 and 128 are stored in EEPROM only.

*Attention*:
- An SGP command on such a parameter will always store it permanently and no extra STGP command is needed.
- Take care when changing these parameters, and use the appropriate functions of the TMCL-IDE to do it in an interactive way!

**MEANING OF THE LETTERS IN COLUMN *ACCESS***

| Access Type | Related Command(s) | Description |
|---|---|---|
| R | GGP | Parameter readable |
| W | SGP, AGP | Parameter writable |
| E | STGP, RSGP | Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STGP command and also explicitly restored (copied back from EEPROM into RAM) using RSGP. |

**GLOBAL PARAMETERS (BANK 0)**

| Number | Parameter | Description | Range | Access |
|---|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | 0… 255 | RWE |
| 65 | RS485 baud rate*) | 0 9600 baud *Default*<br>1 14400 baud<br>2 19200 baud<br>3 28800 baud<br>4 38400 baud<br>5 57600 baud<br>6 76800 baud *Not supported by Windows!*<br>7 115200 baud<br>8 230400 baud *)<br>9 250000 baud *) *Not supported by Windows!*<br>10 500000 baud *) *Not supported by Windows!*<br>11 1000000 baud *) *Not supported by Windows!*<br><br>*) *for hardware version V1.3, only! Not supported with hardware version V1.2* | 0… 11 | RWE |

| Number | Parameter | Description | Range | Access |
|---|---|---|---|---|
| 66 | Serial address | The module (target) address (default: 1)<br>*Please note: address 0 has a special meaning. This address is accepted by all modules regardless of their particular address setting. Sending a command with this address might be useful in case a module has been set to an unknown address.* | 0… 255 | RWE |
| 67 | ASCII mode | Configure the TMCL ASCII interface:<br>Bit 0: 0 – start up in binary (normal) mode<br>1 – start up in ASCII mode<br>Bits 4 and 5:<br>00 – echo back each character<br>01 – echo back complete command<br>10 – do not send echo, only send command reply | | RWE |
| 68 | Serial heartbeat | Serial heartbeat for the RS485 interface. If this time limit is up and no further command is noticed the motor will be stopped.<br>0 – parameter is disabled | [ms] | RWE |
| 69 | CAN bit rate | 2 20kBit/s<br>3 50kBit/s<br>4 100kBit/s<br>5 125kBit/s<br>6 250kBit/s<br>7 500kBit/s<br>8 1000kBit/s *Default* | 2… 8 | RWE |
| 70 | CAN reply ID | The CAN ID for replies from the board (default: 2) | 0… 7ff | RWE |
| 71 | CAN ID | The module (target) address for CAN (default: 1) | 0… 7ff | RWE |
| 73 | Configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. | 0/1 | RWE |
| 75 | Telegram pause time | Pause time before the reply via RS485 is sent. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin). For CAN interface this parameter has no effect! | 0… 255 | RWE |
| 76 | Serial host address | Host address used in the reply telegrams sent back via RS485. | 0… 255 | RWE |
| 77 | Auto start mode | 0: Do not start TMCL application after power up (default).<br>1: Start TMCL application automatically after power up. | 0/1 | RWE |
| 79 | End switch polarity | 0: normal polarity<br>1: reverse polarity | 0/1 | RWE |

| Number | Parameter | Description | Range | Access |
|---|---|---|---|---|
| 81 | TMCL code protection | Protect a TMCL program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>1 – protection against overwriting<br>1 – protection against disassembling and overwriting<br>*If you switch off the protection against disassembling, the program will be erased first!*<br>*Changing this value from 1 or 3 to 0 or 2, the TMCL program will be wiped off.* | 0,1,2,3 | RWE |
| 83 | CAN secondary address | Second CAN ID for the module. Switched off when set to zero. This is the group or broadcast address of the module. Using this address a single command e.g. ROR or MVP sent by the master is sufficient in order to initiate a movement of several (group) or even all (broadcast) modules connected to one bus. The first CAN address (71) might then still be used to set parameters of the modules individually.<br>In order to avoid bus collisions, the module will **not** sent a reply for commands with this address. | 0… 7ff | RWE |
| 84 | Coordinate storage | 0 – coordinates are stored in the RAM only (but can<br>be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM<br>only | 0/1 | RWE |
| 85 | Do not restore user variables | 0 – user variables are restored (default)<br>1 – user variables are not restored (default) | 0/1 | RWE |
| 87 | Serial secondary address | Second module (target) address. This is the group or broadcast address of the module. Using this address a single command e.g. ROR or MVP sent by the master is sufficient in order to initiate a movement of several (group) or even all (broadcast) modules connected to one bus. The first serial address (66) might then still be used to set parameters of the modules individually.<br>In order to avoid bus collisions, the module will **not** sent a reply for commands with this address. | 0… 255 | RWE |
| 128 | TMCL application status | 0 –stop<br>1 – run<br>1 – step<br>3 – reset | 0… 3 | R |
| 129 | Download mode | 0 – normal mode<br>1 – download mode | 0/1 | R |
| 130 | TMCL program counter | The index of the currently executed TMCL instruction. | | R |

| Number | Parameter | Description | Range | Access |
|---|---|---|---|---|
| 132 | Tick timer | A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value. | 0… $2^{32}$ | RW |
| 133 | Random number | Choose a random number. | 0… 2147483647 | RW |

*) With most RS485 converters that can be attached to the COM port of a PC the data direction is controlled by the RTS pin of the COM port. Please note that this will only work with Windows 2000, Windows XP or Windows NT4, not with Windows 95, Windows 98 or Windows ME (due to a bug in these operating systems). Another problem is that Windows 2000/XP/NT4 switches the direction back to *receive* too late. To overcome this problem, set the *telegram pause time* (global parameter #75) of the module to 15 (or more if needed) by issuing an *SGP 75, 0, 15* command in direct mode. The parameter will automatically be stored in the configuration EEPROM.

## 5.2 Bank 1

The global parameter bank 1 is normally not available. It may be used for customer specific extensions of the firmware. Together with user definable commands these variables form the interface between extensions of the firmware (written in C) and TMCL applications.

## 5.3 Bank 2

Bank 2 contains general purpose 32 bit variables for the use in TMCL applications. They are located in RAM and the first 56 variables can be stored permanently in EEPROM, also. After booting, their values are automatically restored to the RAM. Up to 256 user variables are available.

**MEANING OF THE LETTERS IN COLUMN *ACCESS***

| Access Type | Related Command(s) | Description |
|---|---|---|
| R | GGP | Parameter readable |
| W | SGP, AGP | Parameter writable |
| E | STGP, RSGP | Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STGP command and also explicitly restored (copied back from EEPROM into RAM) using RSGP. |

**GENERAL PURPOSE VARIABLES FOR TMCL APPLICATIONS (BANK 2)**

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 0… 55 | general purpose variables #0… #55 | for use in TMCL applications | $-2^{31}$… $+2^{31}$ | RWE |
| 56… 255 | general purpose variables #56… #255 | for use in TMCL applications | $-2^{31}$… $+2^{31}$ | RW |

## 5.4 Bank 3

Bank 3 contains interrupt parameters. Some interrupts need configuration (e.g. the timer interval of a timer interrupt). This can be done using the SGP commands with parameter bank 3 (SGP <type>, 3, <value>). ***The parameter number defines the priority of an interrupt. Interrupts with a lower number have a higher priority.***

**MEANING OF THE LETTERS IN COLUMN *ACCESS***

| Access type | Related command(s) | Description |
|---|---|---|
| R | GGP | Parameter readable |
| W | SGP, AGP | Parameter writable |

**INTERRUPT PARAMETERS (BANK 3)**

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 0 | Timer 0 period (ms) | Time between two interrupts (ms) | 0… 4.294.967.295 [ms] | RW |
| 1 | Timer 1 period (ms) | Time between two interrupts (ms) | 0… 4.294.967.295 [ms] | RW |
| 2 | Timer 2 period (ms) | Time between two interrupts (ms) | 0… 4.294.967.295 [ms] | RW |
| 27 | Stop left 0 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |
| 28 | Stop right 0 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |
| 39 | Input 0 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |
| 40 | Input 1 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |
| 41 | Input 2 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |
| 42 | Input 3 trigger transition | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RW |

# 6 TMCL Programming Techniques and Structure

## 6.1 Initialization

The first task in a TMCL program (like in other programs also) is to initialize all parameters where different values than the default values are necessary. For this purpose, SAP and SGP commands are used.

## 6.2 Main Loop

Embedded systems normally use a main loop that runs infinitely. This is also the case in a TMCL application that is running stand alone. Normally the auto start mode of the module should be turned on. After power up, the module then starts the TMCL program, which first does all necessary initializations and then enters the main loop, which does all necessary tasks end never ends (only when the module is powered off or reset).

*There are exceptions to this, e.g. when TMCL™ routines are called from a host in direct mode.*

So most (but not all) stand alone TMCL programs look like this:

```
//Initialization
      SAP 4, 0, 500  //define max. positioning speed
      SAP 5, 0, 100  //define max. acceleration

MainLoop:
      //do something, in this example just running between two positions
      MVP ABS, 0, 5000
      WAIT POS, 0, 0
      MVP ABS, 0, 0
      WAIT POS, 0, 0
      JA MainLoop       //end of the main loop => run infinitely
```

## 6.3 Using Symbolic Constants

To make your program better readable and understandable, symbolic constants should be taken for all important numerical values that are used in the program. The TMCL-IDE provides an include file with symbolic names for all important axis parameters and global parameters.

Example:

```
//Define some constants
#include TMCLParam.tmc
MaxSpeed = 500
MaxAcc = 100
Position0 = 0
Position1 = 5000

//Initialization
      SAP APMaxPositioningSpeed, Motor0, MaxSpeed
      SAP APMaxAcceleration, Motor0, MaxAcc

MainLoop:
      MVP ABS, Motor0, Position1
      WAIT POS, Motor0, 0
      MVP ABS, Motor0, Position0
      WAIT POS, Motor0, 0
      JA MainLoop
```

*Just have a look at the file TMCLParam.tmc provided with the TMCL-IDE. It contains symbolic constants that define all important parameter numbers.*

Using constants for other values makes it easier to change them when they are used more than once in a program. You can change the definition of the constant and do not have to change all occurrences of it in your program.

## 6.4 Using Variables

The *User Variables* can be used if variables are needed in your program. They can store temporary values. The commands SGP, GGP and AGP are used to work with user variables:

*SGP* is used to set a variable to a constant value (e.g. during initialization phase).

*GGP* is used to read the contents of a user variable and to copy it to the accumulator register for further usage.

*AGP* can be used to copy the contents of the accumulator register to a user variable, e.g. to store the result of a calculation.

Example:

```
MyVariable = 42
      //Use a symbolic name for the user variable
      //(This makes the program better readable and understandable.)

SGP MyVariable, 2, 1234   //Initialize the variable with the value 1234
…
…
GGP MyVariable, 2         //Copy the contents of the variable to the
accumulator register
CALC MUL, 2                //Multiply accumulator register with two
AAP MyVariable, 2         //Store contents of the accumulator register to the
variable
…
…
```

Furthermore, these variables can provide a powerful way of communication between a TMCL program running on a module and a host. The host can change a variable by issuing a direct mode SGP command (remember that while a TMCL program is running direct mode commands can still be executed, without interfering with the running program). If the TMCL program polls this variable regularly it can react on such changes of its contents.

The host can also poll a variable using GGP in direct mode and see if it has been changed by the TMCL program.

## 6.5 Using Subroutines

The *CSUB* and *RSUB* commands provide a mechanism for using subroutines. The *CSUB* command branches to the given label. When an *RSUB* command is executed the control goes back to the command that follows the *CSUB* command that called the subroutine.

This mechanism can also be nested. From a subroutine called by a *CSUB* command other subroutines can be called. In the current version of TMCL eight levels of nested subroutine calls are allowed.

## 6.6 Mixing Direct Mode and Standalone Mode

Direct mode and standalone mode can also be mixed. When a TMCL program is being executed in standalone mode, direct mode commands are also processed (and they do not disturb the flow of the program running in standalone mode). So, it is also possible to query e.g. the actual position of the motor in direct mode while a TMCL program is running.

Communication between a program running in standalone mode and a host can be done using the TMCL user variables. The host can then change the value of a user variable (using a direct mode SGP command) which is regularly polled by the TMCL program (e.g. in its main loop) and so the TMCL™ program can react on such changes. Vice versa, a TMCL program can change a user variable that is polled by the host (using a direct mode GGP command).

A TMCL program can be started by the host using the run command in direct mode. This way, also a set of TMCL routines can be defined that are called by a host. In this case it is recommended to place JA commands at the beginning of the TMCL program that jump to the specific routines. This assures that the entry addresses of the routines will not change even when the TMCL routines are changed (so when changing the TMCL routines the host program does not have to be changed).

Example:

```
//Jump commands to the TMCL™ routines
Func1:      JA Func1Start
Func2:      JA Func2Start
Func3:      JA Func3Start

Func1Start: MVP ABS, 0, 1000
            WAIT POS, 0, 0
            MVP ABS, 0, 0
            WAIT POS, 0, 0
            STOP

Func2Start: ROL 0, 500
            WAIT TICKS, 0, 100
            MST 0
            STOP

Func3Start:
            ROR 0, 1000
            WAIT TICKS, 0, 700
            MST 0
            STOP
```

This example provides three very simple TMCL routines. They can be called from a host by issuing a run command with address 0 to call the first function, or a run command with address 1 to call the second function, or a run command with address 2 to call the third function. You can see the addresses of the TMCL labels (that are needed for the run commands) by using the *Generate symbol file* function of the TMCL-IDE.

*Please refer to the TMCL-IDE User Manual for further information about the TMCL-IDE.*

# 7 Life Support Policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

© TRINAMIC Motion Control GmbH & Co. KG 2013

Information given in this data sheet is believed to be accurate and reliable. However neither responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties, which may result from its use.

Specifications are subject to change without notice.

# 8 Revision History

## 8.1 Firmware Revision

| Version | Date | Description |
|---|---|---|
| 1.18 | 2012-MAY-06 | Release |
| 1.19 | 2012-JUL-25 | Global parameter 79 added |
| 1.20 | 2012-OKT-04 | - Global parameter 87 (secondary address for RS232/RS485) added.<br>- Reference search: the last position before setting the counter to zero can be read out with axis parameter 197. |
| 1.21 | 2012-NOV-16 | Parameter VSENSE set to 1. |
| 1.22 | 2013-JAN-21 | - Maximum read number of encoder increased.<br>- Additional functions of axis parameter 193 (reference search mode):<br>▪ Add 128 to a value for inverting the home switch (interesting for mode 5… 8).<br>▪ Add 64 to a value for driving the right instead of the left reference switch (interesting for mode 1… 4). |
| 1.23 | 2013-FEB-05 | Reference search modes corrected. Mode 7 and mode 8: end switches are always deactivated. |
| 1.24 | 2013-FEB-20 | No changes related to the TMCM-1140 |
| 1.25 | 2013-AUG-30 | No changes related to the TMCM-1140 |
| 1.26 | 2013-AUG-30 | No changes related to the TMCM-1140 |
| 1.27 | 2013-AUG-30 | Problem with magnetic encoder fixed |

## 8.2 Document Revision

| Version | Date | Author | Description |
|---|---|---|---|
| 1.00 | 2012-JUN-20 | SD | First version |
| 1.01 | 2012-JUL-27 | SD | - SIO command description completed.<br>- Axis parameter 141 deleted.<br>- Global parameter 79 added. |
| 1.02 | 2013-MAR-26 | SD | - Interrupt description completed.<br>- Axis parameter 218 corrected.<br>- Global parameters 84 and 85 added.<br>- GIO command description and SIO command description updated.<br>Names of inputs changed:<br>~~AIN_0~~ IN_0<br>~~IN_0~~ IN_1<br>~~IN_1~~ IN_2<br>~~IN_2~~ IN_3<br>- Global parameter 67 (ASCII) added.<br>- Global parameter 87 (secondary address for RS485) added.<br>- Reference search: the last position before setting the counter to zero can be read out with axis parameter 197.<br>- Axis parameter 193: new functions added. |
| 1.03 | 2013 | JP | Revision History updated |
| 1.04 | 2015-JAN-05 | GE | - Hardware version V1.3 (pictures updated)<br>- Reference search mode description (4.3) corrected / updated + clarified<br>- Encoder prescaler description (4.4) clarified / extended<br>- Minor changes / clarifications |

# 9 References

| | |
|---|---|
| [TMCM-1140] | TMCM-1140 Hardware Manual |
| [TMC262] | TMC262 Datasheet |
| [TMC429] | TMC429 Datasheet |
| [TMCL-IDE] | TMCL-IDE User Manual |

Please refer to www.trinamic.com.